RADC-TR-88-324, Vol V (of nine)
Interim Report
March 1989

NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 Building an Intelligent Assistant: The Acquisition, Integration, and Maintenance of Complex Distributed Tasks

Syracuse University

V. Lesser, W.B. Croft, B. Woolf

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-88-324, Vol V (of nine) has been reviewed and is approved for publication.

APPROVED:

DOUGLAS A. WHITE
Project Engineer

APPROVED:

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:

IGOR G. PLONISCH
Directorate of Plans & Programs

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-324, Vol V (of nine)
6a. NAME OF PERFORMING ORGANIZATION Northeast Artificial Intelligence Consortium (NAIC); 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)
6c. ADDRESS (City, State, and ZIP Code) 409 Link Hall Syracuse University Syracuse NY 13244-1240	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center; 8b. OFFICE SYMBOL (If applicable) COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0008
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61102F	2304	J5	01
62702F	5581	27	13

11. TITLE (Include Security Classification)
NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT 1987 Building an Intelligence Assistant: The Acquisition, Integration, and Maintenance of Complex Distributed Tasks

12. PERSONAL AUTHOR(S)
V. Lesser, W. B. Croft, B. Woolf

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM Dec 86 TO Dec 87	March 1989	588

16. SUPPLEMENTARY NOTATION
This effort was performed as a subcontract by the University of Massachusetts to Syracuse University, Office of Sponsored Programs.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Intelligent Interfaces, Planning, Plan Recognition, Intelligence Computer-Aided Instruction
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the third year of the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance distributed AI for communications system control, automatic photo interpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system. The specific topic for this volume is the development of intelligent interfaces to support cooperating computer users in their interactions with a computer.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL DOUGLAS A. WHITE	22b. TELEPHONE (Include Area Code) (315) 330-4833; 22c. OFFICE SYMBOL RADC/COES

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

5 Building an Intelligent Assistant: The Acquisition, Integration, and Maintenance of Complex Distributed Tasks

Victor Lesser
W. Bruce Croft
Beverly Woolf

1987 Annual Report to RADC

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Contents

Chapter 5

5.1 Executive Summary

The NAIC-affiliated AI group at the University of Massachusetts has completed a number of projects this year in the area of intelligent interfaces. These projects have made important contributions towards our goal of supporting cooperating users in their interaction with a computer and tutoring them about the expertise residing in the knowledge bases they are using. Our research has concentrated on issues in planning, plan recognition, knowledge representation, knowledge acquisition, cooperative and distributed problem solving, and intelligent tutoring. Advances in each of these areas are essential for a system to be able to understand the goals of a user, relate these goals to other users' goals, formulate plans to accomplish the goals, and successfully execute these plans in interactive environments. We have emphasized the importance of techniques that can deal with open-ended domains where the actions of agents are not completely predictable but are generally purposeful. We feel that domains with these characteristics are found in many important applications.

In the area of planning and plan recognition we have completed implementation of a new plan recognition formalism (GRAPPLE) that provides a hierarchy of procedural descriptions or plans specifying typical user tasks, goals, and sequences of actions to accomplish goals. Included within this formalism is a framework for meta-plans and first-principles knowledge.

We have also built a testbed system (POLYMER) for developing replanning, negotiation, and knowledge acquisition techniques. Exceptions that occur during interactive plan execution are handled by constructing explanations from the current knowledge base and then using this structure as the basis for negotiation between affected agents. This approach can be viewed as a special case of explanation-based learning. Another important aspect of knowledge acquisition is the design of interfaces based on cognitive models of the way people view their activities. A prototype system based on this idea has been started this year. We have also started research on how cooperative agents can negotiate to resolve conflicts in their viewpoints.

Several dissertations were completed this year. One was a Ph.D. thesis on knowledge acquisition. This thesis developed the idea of assimilating new knowledge with similar knowledge already in the knowledge base. An M.S. project was completed that developed a scheduling system for representing and reasoning about time. Another Ph.D. thesis, that produced an object-oriented graphic interface for decision support, is in its final stages of preparation.

Also this year, we designed and began implementation of a question/answering system for qualitative reasoning about complex engineering systems. The system allows a user to design and build environments to teach concepts such as statics, thermodynamics, and kinematics. Building these interfaces required extensive knowledge engineering with educators, psychologists, and physicists.

In sum, this year we have seen the completion of several projects, continued development of others, and the beginnings of still others in the area of intelligent interfaces. This work represents a major advance in a number of areas with regard to the previous year.

5.2 Introduction

The NAIC-affiliated AI group at the University of Massachusetts has been working on development of interfaces that support cooperating computer users in their interactions with a computer. These interfaces contain knowledge about typical methods used by people to achieve tasks as well as knowledge to recognize their plans, help them complete their task, and provide explanation in support of their activities.

To be approachable and informative, an intelligent interface must be able to understand what the user intends and to rectify the user's misunderstandings gracefully. It must both justify its performance and facilitate its own modification. Preferably such an interface should have command of several communications media, including natural language, programming languages, and graphics, and use these media where they are most appropriate. Needless to say, no current interfaces are capable of this range of communicative skill.

The mechanisms we have built facilitate a user's ability to interact naturally with a system and they improve the computer's ability to describe its own actions and decisions in a clear and user-centered manner. We have concentrated on several major areas of intelligent interfaces, including planning, plan recognition, knowledge representation/acquisition, and cooperative problem solving. This report describes work in each of these four areas.

5.3 Planning and Plan Recognition

Planning is the process whereby a system uses an explicit list of preconditions and goals to execute actions. Planners can take many forms, e.g., they can be hierarchical, script-based or opportunistic. We are developing sophisticated planning systems to dynamically recognize and deal with exceptions from expected plans [2] and to specify complex exception-handling strategies through meta-plans [8].

Plans specify a hierarchical relationship between data and more abstract views of this data. Although the form and specification of plans vary with the application domain, plans are composed of sets or sequences of subplans. For example, a plan for processing a form is composed of steps for filling the form out and then sending it on to the appropriate office. In the case of monitoring a vehicle, a plan might be composed of subplans which represent sets or sequences of radio and radar emissions that identify the vehicle and its purpose.

On the other hand, plan recognition is the interpretation of data in terms of instances of particular plans. As such, it is distinguished by reliance on incomplete and uncertain

knowledge. It involves the interpretation of sets of subplan instances as (perhaps partial) instances of more abstract plans.

Plan recognition is a complex and uncertain process. Often there are multiple, ambiguous interpretations for each subplan or sequence of subplans. Data may also be missing, uncertain, and/or incorrect.

The prototypical example of plan recognition is the interpretation of a series of actions as part of some overall task. This ability is relevant to natural language understanding and computerized intelligent assistants. For example, vehicle monitoring and related situation assessment problems may also be treated as plan recognition problems. Here, the "plans" represent vehicle movements or missions and are composed of characteristic sequences of sensor data rather than "actions." In all, the goal is to form a higher-level, more abstract view of the data. Stated another way, the goal is to provide an appropriate context within which to understand the data. A large range of interpretation and situation assessment problems can be viewed as plan recognition problems.

5.4 A Unified Planning/Plan Recognition Framework

We see planning as a cooperative effort between user and machine. Thus, the individual's task often cannot be fully automated. Frequently the user and machine must *cooperate* to achieve a common goal, which is a task within a plan. We have begun work on POLYMER, a planning system designed to support cooperative user activities [12]. We intend that the planner:

- **allow the user to initiate planner activity** and to invoke the planner for assistance;
- **be interactive** or rely on the user to supply control decisions as well as provide missing information necessary to continue the planning process;
- **not be a stand-alone system**, but rather accept salient information that guides the planning process and imposes constraints on further development of a plan.

Realistically, exceptions and interruptions during the execution of a plan are common occurrences, and an "intelligent assistant" should react to new information as it becomes available during plan construction and execution. We are building a system that finds explanations for exceptions and adjusts its plans to the understood 'exception' (see next section). This exception-handler system will be part of the POLYMER system.

The basic cycle of POLYMER for a single user is as follows:

1. A goal is posted by the application (upon encountering a user action).
2. The planner expands the goal into a partial ordering of activities and subgoals, constructing the set of actions which can occur next in the form of an *expected-actions-list*.
3. The execution monitor selects an action from the *expected-actions-list* and sends a message to an active-object (to the user if it is an action which must be performed by the user, otherwise to a tool object).

4. The execution monitor compares the actual action taken with the set of actions on the *expected-actions-list*, and if a match is not found, the exception-handling mechanism is invoked.

5. Once an explanation has been negotiated successfully (or if a match was found from step 4 above), control is returned either to: a) Step 3 if more actions remain on the *expected-actions-list*, or b) Step 2 otherwise.

This year we have designed and implemented the knowledge representation, a planner, and an execution monitor for the unified planning/plan recognition system [12]. The key issues have been the incorporation of an interactive plan/execution cycle and the use of "world" facilities in KEE for dependency-directed backtracking, constraint propagation and protection interval violation detection. We have begun the definition of a communication protocol between cooperating agents.

The object-management subsystem of POLYMER contains detailed knowledge about the domain activities, the objects they create and manipulate, and the people or "agents" who are responsible for their execution. The activity description language used by POLYMER is based on formalisms used by other planning systems [3,6,14]. The representation provides mechanisms to express causality and includes a general looping mechanism for expressing different forms of iteration. The details of the POLYMER planner and OMS can be found in [12].

A crucial feature of the project is the emphasis on flexible execution and planning achieved through powerful exception-handling techniques (see next section). We have begun implementation of this exception-handling facility as a component of POLYMER.

5.4.1 Exception Handling

A new Ph.D. dissertation has been started which is intended to handle cases when an exception is presented to a system's existing plans [2,1]. The motivating force for this work is the observation that the real world often does not operate in a "typical" fashion. All possible ways of performing a task goal cannot be anticipated, neither can unexpected contingencies be predicted. Current planners are too "breakable" in the face of unexpected events.

In addition, intelligent agents are often important parties in plan execution and sources of knowledge for refining an incompletely specified domain. Similarly, agents perform purposeful actions; their behavior is seldom random. Previous systems have not tried to understand "erroneous" agent actions within a planning framework. Replanning is often a reactionary approach that is not sufficient for sophisticated explanation of unusual occurrences and the corresponding modification of plans.

In response to these characteristics, we are building a "flexible" intelligent assistant to help agents perform tasks. The system should go beyond "reactionary" approaches when encountering unexpected events or states, and should attempt to understand the possible intent behind exceptions and how these exceptions might be incorporated into a consistent plan. This work constitutes an attempt to bridge the gap between the knowledge used to guide system behavior, and that used to guide the agents' behavior. The eventual goal is

to have the system's model of the domain more closely approximate the intelligent agents' model of the domain.

The overall goal of this work is to develop a planning architecture that is robust when encountering unanticipated contingencies. In other words, our planning system will "reason" about exceptional occurrences as they arise, and will include them as part of an evolving plan whenever possible. In addition, we hope to show that future system performance can benefit through exception handling, specifically, that the system can acquire new knowledge about performing domain tasks as the result of an exception.

The basic notion guiding our approach is to use the class of a detected exception together with a heuristic determination of user intent, to select among a set of strategies which attempt to discover a role for the exception in the current plan, if one exists. Otherwise, the system will attempt to negotiate directly with responsible agents in an effort to require additional explanatory information, or replan if necessary.

The system we are building, called SPANDEX, is based on POLYMER and includes additional modules to address exception handling. Exceptions are detected by the *execution monitor* and classified by the *exception classifier*. Violations in the plan caused by the introduction of an exception are computed by the *plan critic*. Exceptions generated by unknown agents (generated by *world* in the diagram) are handled by the *replanner*. The replanning approach we have adopted is similar to that of [15], where one or more of a set of general replanning actions is invoked in response to a particular type of problem introduced into a plan by an exceptional occurrence. For interactive planning, we extend the set of general replanning actions to include the insertion of a new goal into the plan.

Our goal is to show that this approach produces a planning system that is less brittle and more efficient than previous planners which adopt a simpler replanning approach when encountering exceptional occurrences. Finally, we hope to demonstrate that this approach produces a system which "learns" about alternative ways to complete task goals, and is able to use this new knowledge during future planning activities.

5.4.2 The GRAPPLE Plan Recognition System

This year we implemented a second generation intelligent interface, GRAPPLE (Goal Recognition and Planning Environment). This system was discussed in depth in Section 5 of the RADC NAIC 1986 yearly report. It is also detailed in [9]. It will only be summarized here.

GRAPPLE incorporates state-based information and uses meta-plans and reasoning from first principles. It follows the classical planning formalism: A *goal* is specified as a partial state of the semantic database. A goal can be decomposed into *subgoals*, each of which also is expressed as a semantic database state specification. Achievement of all the subgoals, along with the posting of the *effects* of the plan, should lead to satisfaction of the goal of the plan. *Effects* can be expressed in high-level as well as primitive plans, allowing for the expression of complex semantic changes to the semantic database.

A state-based approach to plan representation provides the system more modularity. For example, in the software development environment, if one of the subgoals for a plan is to *have-more-disk-space*, a number of plans may be retrieved that achieve this subgoal; for instance: *delete-a-file*, *purge-directory*, and *increase-quota*. The multiple possible plans need not be

specified statically; they can be determined dynamically in order to exploit the rich sources of contextual knowledge at runtime. Representing goals as states in GRAPPLE also allows the interface to avoid a potentially redundant execution of a plan. If a plan has a subgoal which is already satisfied, then no plan need be executed to achieve the subgoal. The overall ordering of the plans that can achieve subgoals of a complex plan is determined dynamically by monitoring the satisfaction of *preconditions*. The state-based approach thus allows for the easy addition and removal of plan definitions from the plan library, without necessitating a recompilation of all the plans and their subgoals.

5.4.3 Meta-Plans and First-Principles Knowledge

Two problems arise due to the limits in the representational adequacy of existing hierarchical plan formalisms. The first problem concerns the adequacy of operator definition. The difficulties here include capturing such relevant domain knowledge as how to recover when an operator fails or when to use special case operators. We provide a solution based upon defining *meta-plans* that dynamically transform plans. The second problem lies in the scope and role of the domain state schema. Schema can be extended to encompass a deep model of programming process knowledge. *Non-monotonic reasoning* and *first-principles knowledge* are used to compensate for partial knowledge of the extended state.

We have demonstrated both these features in our use of the GRAPPLE planning formalism to describe a plan-based approach to the process of programming [8]. The types of support that can be provided include generation of agendas and summaries of process status, detection and correction of process errors, and cooperative automation of process activities. In this plan-based approach, knowledge of programming processes are expressed as operators defining the legal actions of programming, together with a state schema defining the predicates that describe the state of the programming world. A plan is a partial order of operators (with all variables bound) that achieves a goal given an initial state of the world. The algorithms for monitoring programmer actions are the algorithms of plan recognition, where a plan to achieve some goal is identified incrementally from sequences of actions performed by the programmer. The algorithms for carrying out a programmer goal are the algorithms of planning, where a partial order of operators is generated to achieve the stated goal.

The success of a plan-based approach is dependent on capturing knowledge of the complex programming process in a planning formalism. We have shown that two techniques (meta-plans and non-monotonic reasoning) can be used to significantly extend representational power. We have looked at what programming goals are, how individual tools are used, when special cases arise, how to recover from different types of failures, and what first-principles knowledge is relevant.

The programming process laws are captured in axioms applying to an extended domain state, and default rules compensate for the fact that the extended state may be incomplete. The explicit expression of programming process laws allows reasoning from *first principles*. That is, no set of rules specifically addresses legal choices of baselines as an independent issue. Rather, there are rules that relate this choice to a deeper model involving specifications, and additional rules that allow reasoning within this deeper model. The nature of the default rules determines the degree of certainty in this reasoning; with a suitable set of default rules, the

interface will occasionally draw faulty (but correctable) conclusions, as a result of reasoning independently from the programmer with incomplete information.

Transformations are operations on some world state; in this case, the world state is the plan network. Therefore, such transformations can be formalized as meta-operators and synthesized into meta-plans. This approach has the desired generality; it also adds to the role of meta-plans, which have previously been used to implement control strategies and capture domain-independent knowledge. The primary advantage of meta-plans is the power of having expressive generality in a single formalism, as compared with a collection of ad hoc operator language extensions for each encountered exception. Any aspect of an operator definition (such as preconditions, subgoals, constraints, or effects), as well as any aspect of an operator instance (such as bindings of variables or ancestor operator instances) can be accessed or modified. The transformational approach also addresses practical problems in providing a complete library of operators. Because knowledge of exceptions is partitioned from knowledge of normal cases, the two issues can be tackled separately. The process of writing operators is further improved because multiple transformations can apply to a single operator, thereby preventing combinatorial explosion in the numbers of operators.

5.5 Knowledge Acquisition

Though planners and plan recognition systems have been around for more than a decade, few planners are in use outside research laboratories. This is because of the high cost (time and money) of building customized plan libraries and the difficulties in updating plan libraries. A crucial problem here is the discrepancy in background and expertise of the domain expert, who might be a secretary, clerk, bookkeeper, or manager, and the knowledge engineer. Frequently, the communication between these two is distorted and obstructed. Additionally, constant changes in the tasks, task assignation, and general organizational structures require updates and changes in the plan libraries. In such cases, programmers and knowledge engineers have to be called upon to modify the system.

We need solutions that will make the benefits of planners accessible to applications in the real world. These solutions include knowledge acquisition systems that can facilitate codification of human expert knowledge into the knowledge base of a system. We have worked on two such systems this year, both Ph.D dissertations, and describe them in this section.

5.5.1 Dialogue Mechanism for Domain Knowledge Acquisition

This year a Ph.D. dissertation was completed that performed knowledge acquisition by engaging an expert in a dialogue about which of several interpretations of new knowledge are intended to be included in an existing knowledge base [11]. This work was described in detail in Section 5 of the RADC NAIC 1986 yearly report. It will only be summarized here.

Knowledge acquisition requires an understanding of how information to be incorporated into the system corresponds to information already known by the system. We built a system, called K^nAc that modifies an existing knowledge base by using heuristic knowledge about the

knowledge acquisition process to anticipate modifications to the existing entity descriptions. These anticipated modifications, or *expectations*, are used to provide a context in which to assimilate the new domain information.

An often overlooked aspect of the knowledge acquisition process is the assimilation of information presented by the domain expert into an existing knowledge base. The knowledge engineer's task is modification of the expert system's knowledge base so as to reflect the domain expert's knowledge. To a large extent, this knowledge acquisition task may be viewed as a recognition problem. All of the problems facing other recognition systems are present here, including: noisy data (i.e., incomplete or inaccurate information), ambiguous interpretations, and the need to produce intermediate results before all the data is available. Thus, a significant portion of the interactive knowledge acquisition task is a matching problem: How does the expert's description of the domain correlate with the description contained in the knowledge base? How should the knowledge base be modified based on new information from the expert? What should be done when the expert's description differs from the existing one? $K^{n}{}_{A}C$ implements this knowledge assimilation approach to knowledge acquisition.

5.5.2 An Acquisition Language for Specifying Constraints

We have begun work on a Ph.D. dissertation in the area of knowledge acquisition [13]. This system will update specification of plans through direct manipulation of interface icons and use of a visual programming language.

This research work takes a Cognitive engineering approach to the problem of designing an interface for the acquisition of planning/plan recognition data. Cognitive engineering is the technical application of results and methods from Cognitive Science research, the domain of inquiry that seeks to understand intelligent systems and the nature of intelligence.

We have investigated a theory of the human representation of action-oriented tasks and have applied this theory to AI-planners. The model addresses those syntactic elements that appear in the plan language of the AI-planner, including: the *goal* of the plan, the *effects* of the application, the *preconditions*, the *objects* and the *actors* involved in the *actions*, and the *constraints* among objects and/or objects and actors.

In computational terms, many representations are effective for representing plans. For example, frames are equally good in representing actors, objects and actions. The slots of the frames hold the values of these entities. Production systems have also been used to describe plans. We focus our work on the explicit treatment of constraints, goals and effects, or the formation and representation of any of those three. We take a hybrid approach to the problem of knowledge representation and combine several systems and deal explicitly with goals, constraints and effects.

Our goal is to implement an interface based on a plan specification language that can be handled by novice users, and that requires only a minimal amount of time to be learned. The language needs to be understood by a user who just looks at the code and explores its features. We are attempting to attain this objective by considering the human representation of tasks and applying principles of direct manipulation to the "translation process" from human knowledge to computer information.

All elements of planning languages have to be addressed in this process. Our plan specification language is intended to be a syntactically complete substitute for traditional plan language. Though plan languages of traditional systems differ in some aspects, they share the common concepts described above.

Our plan language addresses constraints among the actions and objects involved in a plan. These constraints differ considerably from static consistency constraints. Constraints in plans are dynamic. They govern temporal bindings and restrictions. Like preconditions they are usually expressed by prepositions. We handle dynamic constraints in a dynamic way by letting users make constraints among elements of the plan rather than by stating these constraints in a global expression.

5.6 Focus-of-Control Issues

Work is in progress as part of a Ph.D. dissertation to investigate evidence-based plan recognition as a focus-of-control mechanism in plan recognition systems [4]. The current work develops a system that will address a frequent problem of existing plan recognition systems, namely their inability to understand and interpret the evidence behind plan recognition decisions. A plan recognition system of any sophistication and generality must be able to meet certain requirements. It must:

1. Evaluate the level of belief and uncertainty in alternative interpretations.
2. Understand the reasons for beliefs.
3. Revise interpretation hypotheses as information accumulates.
4. Handle uncertain and incorrect data.
5. Integrate data from multiple sources.

We are developing a new approach to plan recognition which has these features. The key to this approach is to view plan recognition as a process of *gathering evidence* to *manage uncertainty*. In this way we are able to apply expert-level heuristic control knowledge and to evaluate alternative interpretations.

In the proposed system, data is considered as a source of evidence for the plan hypotheses: when data can be interpreted as part of a hypothesis it provides evidence for that hypothesis. Evidential links are maintained between data and hypotheses the data supports. This provides the system with an explicit representation of the reasons to believe the interpretation hypotheses. The use of an explicit, symbolic representation of evidence is important because it makes possible explicit reasoning about control decisions.

By explicit, symbolic representations for evidence, we simply mean that we maintain explicit links between hypotheses and the reasons we believe the hypotheses. For example, when dealing with vehicular monitoring, sources of evidence include terrain and weather information. Access to detailed information about the evidence makes it possible for us to make use of an important body of expert-level knowledge about the task: the *sources of uncertainty* in

the evidence. In plan recognition, evidence is rarely conclusive. The sources of uncertainty represent the reasons why evidence may fail to support a particular conclusion. For example, acoustic sensor data may fail to support a vehicle because it is actually the result of a sensor malfunction or sensor ghosting. Using this evidence, the control component can reason about the best course of action for the interpretation system to take because it understands the purpose of its actions: to try to resolve the sources of uncertainty in the hypotheses. An independent, explicit representation of the evidence also makes it possible to represent the relations between the hypotheses. Thus, though direct evidence for a hypothesis may not be available, there may be sources of evidence for related hypotheses–like alternatives.

Knowing what evidence supports hypotheses, we can understand the *sources of uncertainty* in the evidence and decide how best to resolve them. When evidence is summarized in numeric degrees of belief, access to this sort of knowledge is lost.

We are building a system in which the system has access to the reasons for its beliefs in order to reason intelligently about control decisions. Evidence provides uncertain support for a hypothesis because there are conditions under which the evidence may fail to support the hypothesis. Numeric rating functions gathered from experts typically summarize just such knowledge—along with *a priori* likelihood judgements. Explicit information about the uncertainties in evidence is a type of knowledge that we feel is very important for the development of more sophisticated AI systems. It allows us to evaluate belief dynamically rather than having to rely on *a priori* likelihoods since it is now possible to enumerate the sources of uncertainty in evidence and to judge their *likelihood* in the current contexts. Control decisions can be directed toward gathering the best evidence to resolve the most critical sources of uncertainty. That is, we can *manage* uncertainty rather than just trying to *resolve* it because we understand exactly what the sources of uncertainty are, which are most critical, and what evidence is best. This applies whether or not the system can interact with its environment to affect the evidence it has available. In any case, the system can direct its actions towards best satisfying its goals given the evidence it has available.

5.7 Cooperative Problem Solving

We are working on several systems to provide an environment for cooperative problem solving between intelligent agents. As part of these systems, intelligent agents (whether human or machine) will be able to exchange reasoning information and knowledge, will be assisted in making cooperative decisions, and will be able to learn new material, input by other agents. These projects are described below.

5.7.1 A Framework for Multi-agent Planning

The POLYMER system described earlier is being used as a testbed to study processes of cooperation and negotiation in a plan-based environment. This research has only just started, but we are concentrating on the use of the explanation structures developed by the exception handler as a basis for negotiation. We expect to implement some version of this this year.

5.7.2 A System for Decision Support

A Ph.D. dissertation which provides graphical aids for decision making is nearing completion. The system was described in detail in Section 5 of the RADC NAIC Annual Report 1986 and will only be summarized here.

ThinkerToy is a graphical environment for modeling decision support problems. It provides a tableau on which problems, such as landscape planning, service scheduling, and statistical analysis can be modeled and analyzed. It uses graphical icons, each associated with physical properties, to replace mathematical relationships and properties. In this system every object is a graphical entity and is directly manipulable. The system allows modeling of scalar objects, arrays, charts, and terrain maps.

5.7.3 Cooperative Problem Solving

Work has begun on a Ph.D dissertation about the interactions between distributed and cooperating knowledge-based systems. In this work, each system is considered an "expert" and each is fairly autonomous but works within a network on a global problem. The areas of expertise of the systems may overlap. The focus of this work is on the negotiation of agreements between experts who propose conflicting solutions or partial solutions.

As there is often no way to make a global evaluation of solutions, the negotiation process must itself be distributed among the experts. This work will evaluate both compromise agreements and agreements that propose novel solutions in which both parties offer proposals that differ significantly from the initial one.

5.7.4 Representing and Reasoning about Time

A master's thesis was completed this year that implemented a system to maintain a personal schedule. The system schedules events, bumps events and reschedules events. It takes into account the ranking of people involved in an event, ongoing projects affected by the event, the status of ongoing projects, distances traveled, preparation time needed, and the personal preferences of the user. The user gives the system an event to schedule, specifying the time period, and the system uses the above constraints to schedule the event.

The system was meant to demonstrate the interaction of various types of considerations that figure into scheduling decisions. It does not optimize the use of any of the data.

5.7.5 Intelligent Tutoring Systems

We have designed general purpose techniques for managing discourse in an intelligent tutor [16]. These techniques are being implemented in structures that dynamically reason about a tutor's response to the student and that customize its generation of examples to the individual student. The structures are flexible, domain-independent, and designed to be rebuilt, i.e., decision points and machine actions are intended to be modified through a visual editor [17,18].

The goal is to have the system respond fluidly to the user and to coordinate its utterances in a more flexible manner than has been required for question/answer or summarization systems.

We have also implemented a mechanism that enables a tutor to retrieve and modify examples, tailoring them to the idiosyncrasies of the student and the curriculum.

The tools enable the tutor to make two kinds of control decisions: high-level decisions determine which strategy to use in responding to the student and low-level choices that determine which example to select next. Both control decisions are based on reasoning about the student's knowledge, the discourse history, and the curriculum.

Decisions at the high level begin or terminate a teaching control strategy; a terminated strategy is replaced with a more appropriate strategy if the student's response pattern has changed [19]. For instance, if a student shows evidence of misunderstanding an unfamiliar situation, the tutor might replace the current strategy with the bridging analogy strategy, which bridges the conceptual gap from the complex situation to a simpler and known one. A library of such strategies will ultimately be available to the tutor along with reasons why it should move from one strategy to another.

Decisions at the low level choose which example to present next. For instance, if the student has made many errors, the tutor might present an example that differs only slightly or perhaps only along a single dimension, from the prior example. On the other hand, another situation might require presentation of a more complex (or more rich or more simple) example. The specification of an example may include questions and problems presented to the student. We intend to generalize the example generation mechanism to attempt to handle all tutor-initiated interactions with the student.

Bibliography

[1] Broverman, C.A., "SPANDEX: An Approach Towards Exception Handling in an Interactive Planning Framework," Ph.D. Dissertation Proposal, November 1987.

[2] Broverman, C.A., Croft, W.B. "Exception Handling During Plan Execution Monitoring," *Proceedings of the Sixth National Conference on Artificial Intelligence*, July 1986, Seattle, WA.

[3] Broverman, C.A., Huff, K.E., Lesser, V.R. "The Role of Plan Recognition in Design of an Intelligent User Interface," *Proceedings of IEEE Conference on Systems, Man, and Cybernetics*, 863-868, 1986.

[4] Carver, N., "Evidence-Based Plan Recognition," Ph.D. dissertation proposal, Nov 1987.

[5] Croft, W.B., Lefkowitz, L.S. "Knowledge-Based Support of Cooperative Activities," *Proceedings of the Hawaii International Conference on System Sciences*, to appear.

[6] Croft, W.B.; Lefkowitz, L.S. "Task Support in an Office System," *ACM Transactions on Office Information Systems*, 2: 197-212; 1984.

[7] "Knowledge Craft Manual Guide," Vax/VMS Version 3.0, Carnegie Group Inc., March 1986.

[8] Huff, K.E., and Lesser, V., "A Plan-Based Intelligent Assistant that Supports the Process of Programming," COINS Tech Report #87-09, September, 1987.

[9] Huff, K.E., and Lesser, V., "The GRAPPLE Plan Formalism," COINS Tech Report, #87-08, April, 1987.

[10] Hayes, P.J. "A Representation for Robot Plans," *Proceedings IJCAI-75*, 181–188, 1975.

[11] Lefkowitz, L. "Knowledge Acquisition Through Anticipation of Modifications," Ph.D. Dissertation, Sept, 1987.

[12] Lefkowitz, L.S. and Croft, W.B. "An Interim Report on the POLYMER Planning System," COINS Technical Report, University of Massachusetts, Amherst, MA. 1987.

[13] Mahling, D. "Dacron: A Plan Specification System for Inexperienced Users, Ph. D. Proposal, to appear.

[14] Wilkins, D.E. "Domain-independent Planning: Representation and Plan Generation," *Artificial Intelligence*, 22: 269-301; 1984.

[15] Wilkins, D.E. "Recovering from Execution Errors in SIPEs," SRI International Technical Report 346, 1985.

[16] Woolf, B., Murray, T., Suthers D., and Schultz, K., "Building Tools for Tutorial Discourse" submitted to *The Journal of Artificial Intelligence in Education.*

[17] Woolf, B., and Cunningham, P., "Multiple Knowledge Sources in Intelligent Tutoring Systems," in *IEEE Expert*, Summer, 1987.

[18] Woolf, B., and Cunningham, P., "Building a Community Memory for Intelligent Tutoring Systems," *Proceedings of the National Association of Artificial Intelligence* (AAAI-87), Morgan Kaufmann, Inc., Los Altos, CA. 1987.

[19] Woolf, B., and Murray, T., A Framework for Representing Tutorial Discourse, *International Joint Conference in Artificial Intelligence (IJCAI-87)*, Morgan Kaufmann, Inc., Los Altos, CA. 1987.

Appendix 5-A

A Plan-Based Intelligent Assistant That Supports the Process of Programming

Karen E. Huff and Victor R. Lesser

COINS Technical Report 87-09
September 1987

Computer and Information Science Department
University of Massachusetts
Amherst, MA. 01003

Abstract: We describe an architecture for providing intelligent assistance to the programmer carrying out the *process* of programming. This architecture, based on an AI *planning* paradigm, can provide both passive and active assistance. Passive assistance, accomplished by *plan recognition*, is used to detect and avert process errors. Active assistance, accomplished by *planning*, is used to automate the programming process. A key issue in achieving appropriate levels of assistance is the ability to capture complex domain knowledge in a planning formalism. We illustrate two limitations in traditional hierarchical formalisms, and present solutions based on the use of *meta-plans* and *non-monotonic reasoning*.

1.0 Introduction

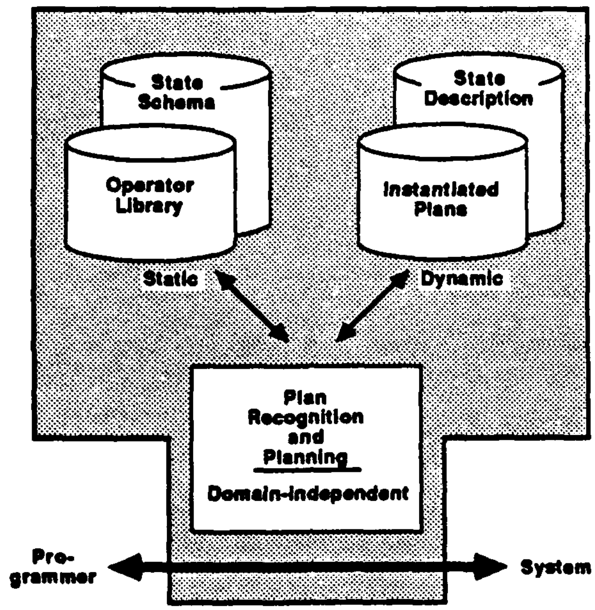
Development environments of today provide little assistance to the programmer with the *process* of programming. The tasks of mapping programming goals into sequences of tool invocations, revising plans when results are not as expected, and performing the required — but mundane — housekeeping chores such as file management are carried out with only rudimentary support. As new techniques are developed to automate some part of the process, new tools are created or existing tools expanded. This leads to a situation where, by definition, the process of programming comprises all the activities that cannot be fully automated.

Even when full automation of the process is precluded, other forms of assistance are still possible. Two approaches, based upon reasoning about the programming process, appear promising. In one case, the programmer retains the initiative for performing the process, issuing commands exactly as at present. A *passive* intelligent assistant *monitors* these actions, measuring them against its (extensive but still incomplete) knowledge of the process. In this mode, many types of process errors could be detected and averted. Such an assistant would be an automated version of a colleague watching over the shoulder of a programmer at work. In the second case, the programmer relinquishes control to the intelligent assistant, specifying only goals to be achieved rather than the detailed commands by which they are to be achieved. Here, an *active* intelligent assistant *plans* and executes a sequence of commands using its knowledge of process actions; since its knowledge is incomplete, the programmer must supply certain decisions which are beyond the scope of the assistant. In this mode, a cooperative automation of the process is achieved.

1.1 Architecture for Intelligent Assistance

We have designed an architecture (diagrammed in Figure 1) that combines both of these forms of assistance in order to provide the programmer a very powerful and flexible support environment. Our approach is based on the use of AI planning techniques, which offer a well-developed framework for reasoning about sequences of actions. Previous applications of planning to software engineering have addressed the plans that underlie programs [9,21]. When applied to the programming process, planning technology represents one possible route towards "process programming" [14], a concept that is the subject of current debate [11]. The GRAPPLE plan-based system that we are currently developing [2,7] builds upon earlier work in intelligent assistant architectures [1,3,5,6].

Figure 1: Architecture for
An Intelligent Assistant



Under a planning paradigm, knowledge of the process is expressed as *operators* defining the legal actions of programming, together with a *state schema* defining the predicates that describe the state of the programming world. A *plan* is a partial order of operators (with all variables bound) that achieves a goal given an initial state of the world. The algorithms for monitoring programmer actions are the algorithms of *plan recognition*, where a plan to achieve some goal is identified incrementally from sequences of actions performed by the programmer. The algorithms for carrying out a programmer goal are the algorithms of *planning*, where a partial order of operators is generated to achieve the stated goal.

A major benefit of this approach is that the intelligent assistant is domain-independent. Changing the library of operators and associated state schema is all that is needed to accommodate alternative programming processes, different toolsets, and project-specific policies. Enlarging the library of operators allows coverage of additional life-cycle phases (and the all-important feedback loops among phases). The intelligent assistant can act at the operating system command level and/or within a complex tool (by considering the functions provided by the tool to be tools themselves.)

```
OPERATOR build
GOAL:       status(?system,built)
PRECOND:    true
SUBGOALS:   created(?system,?baseline)
            status(?system,compiled)
            status(?system,linked)
            status(?system,tested)
EFFECTS:    SET status(?system,built)
```

```
OPERATOR release
GOAL:       status(?system,released)
PRECOND:    status(?system,built)
CONSTR:     current-release(?c)
EFFECTS:    SET status(?system,released)
            DELETE current-release(?c)
            ADD current-release(?system)
            ADD customer-release(?system)
```

```
OPERATOR test
GOAL:       status(?system,tested)
PRECOND:    true
SUBGOALS:   unit-tests(?system,ready)
            unit-tests(?system,run)
EFFECTS:    SET status(?system,tested)
```

```
OPERATOR link
GOAL:       status(?system,linked)
PRECOND:    status(?system,compiled)
EFFECTS:    NEW load-module ?lm
            SET status(?system,linked)
            ADD lm-for(?system,?lm)
            SET time(?system,?time)
```

<u>Relationships</u>

Build satisfies precondition of Release
Link satisfies subgoal of Build
Test satisfies subgoal of Build

Figure 2:
A Partial Library of Simplified Software Process Operators

1.2 Processes as Plans

As a test domain for this intelligent assistant, we are exploring the programming process as it is currently carried out for a traditional programming language such as C, at the command level under an existing operating system (UNIX™), assuming accepted engineering practice (including incremental development, source code control, bug report database, and specialized test suites). A partial library of operators for this domain is sketched in Figure 2; these operators have been simplified for purposes of this example, but serve to indicate the general nature of the approach. The state schema supporting these operators is (partially) sketched in Figure 3, using the entity-relationship model of data [4] as the graphical presentation; relationships and attributes correspond to the logical predicates used in the operator definitions.

The operator definitions follow standard state-based, hierarchical planning approaches [16,18,23]. In a state-based approach, each operator has a *precondition* defining the state that must hold in order for the action to be legal, and a set of *effects* that defines the state changes that result from performing the action. These core clauses are augmented by a *goal*

UNIX is a registered trademark of AT&T Bell Laboratories.

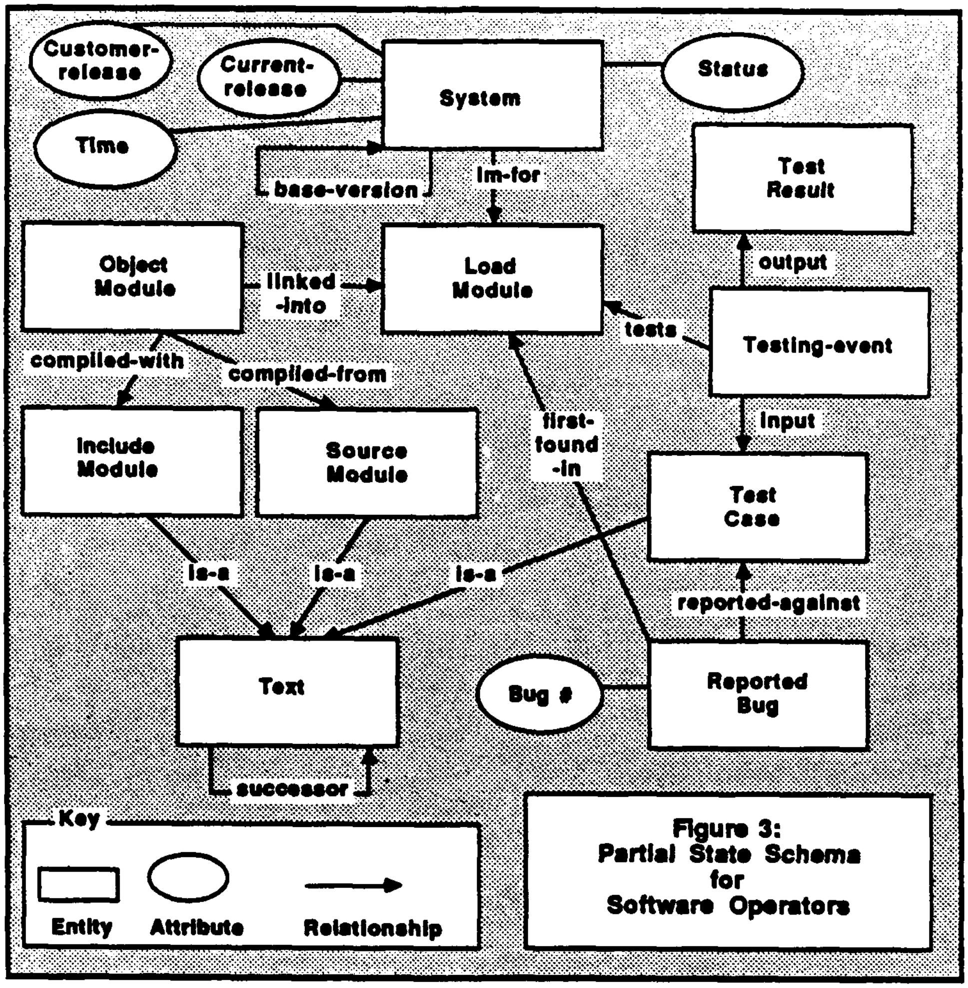
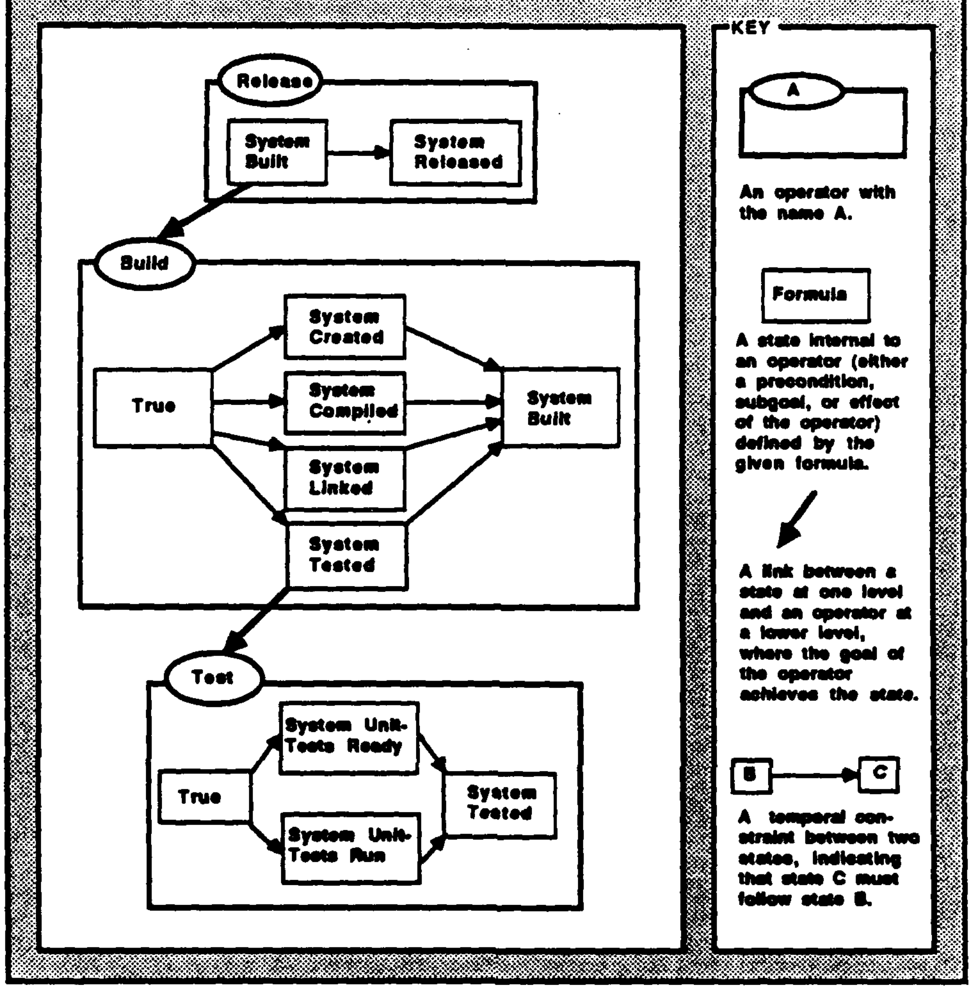


Figure 3: Partial State Schema for Software Operators

clause that defines the principal effects of an action (thus distinguishing them from "side-effects" of the action), and a *constraints* clause that defines restrictions on parameter values. A hierarchical approach supports multiple levels of abstraction in operators by allowing the definition of complex operators having *subgoals* that decompose the operator into simpler subproblems. Primitive operators, which do not have subgoals, correspond to the atomic actions in the domain.

The basic data structure of a planner or plan recognizer is a hierarchical plan network as first developed in [16]. An example of such a network (using some of the operators of Figure 2) is given in Figure 4. There, a vertical slice through a network covering three hierarchical levels is shown, with the highest level at the top of the figure. Downward arrows between levels connect desired states with operators instantiated to achieve them. Such instantiated operators consist of additional states describing the internals of the operator: preconditions, subgoals, and effects. Arrows within levels show how the

FIGURE 4: Example Hierarchical Plan Network



achievement of certain states is partially ordered with respect to time. Some orderings are dictated by the operator definitions: precondition states must always precede subgoals, for example. Other orderings are imposed to resolve interactions between operators that could destroy the plan. Orderings are propagated from level to level, but have been omitted to simplify the figure.

Both planning and plan recognition involve building a complete plan network. This is done by actions such as choosing operators to achieve states, instantiating these operators, and resolving conflicts between newly revealed states and existing states. The strength of a planning approach lies in this ability to handle conflicts that would otherwise destroy a plan. For example, consider a situation where an operator has a two part precondition, requiring that both A and B be true, and the only operator available for achieving B also

achieves NOT A. Any plan that allowed the operator for achieving B to follow the operator for achieving A would fail, due to this interaction of the operators. A viable plan must require that the operator to achieve B precede the operator to achieve A. While reordering operators solves a common type of interaction, other means of conflict resolution have also been developed [16,18,23].

Planning techniques are needed when the chosen problem representation has rules that are not decomposable [13], i.e., when solutions to parallel subproblems cannot be tackled independently and trivially recombined. Some production rule systems assume decomposability, thus avoiding the overhead of dealing with interactions. While such rule systems are effective for providing some types of process automation for programmers [10], interactions such as the precondition example described above cannot be handled. Deleting files is a simple source of interaction; other interactions arise when multiple plans are simultaneously in progress, as is often the case with programming work. For example, when a programmer is both fixing a bug in a customer release and adding new functionality in the latest version, different directories must be used to separate the two working sets of source and object modules.

1.3 *Assistance from a Planning Perspective*

Together, planning and plan recognition make it possible to deliver a broad range of services to the programmer. The planning perspective suggests ways that the specific services can be accomplished:

- **Agendas and Summaries:**
  An agenda is the set of goals yet to be satisfied in a plan, and a summary is the set of goals that have been satisfied. Either can be described at various levels of abstraction, given the hierarchical nature of the plans. Both are "intelligent" in that the system has the knowledge to interpret what they mean: for example, a plan for carrying out an agenda item can be constructed.

- **Error Detection:**
  Three different classes of errors can be handled. Logistical errors represent faulty planning by the programmer. Examples are executing an action before its precondition is met, or undoing a previously satisfied precondition before the relevant action has started. Housekeeping errors represent errors of omission. Examples are keeping files in the "right" directories, checking sources back into a source code control system,

deleting extraneous files, and perhaps adhering to certain project-specific policies. The final class of errors are substantive software process errors: for example, violating constraints in operators such as making the wrong choice of a baseline from which to develop a new system version.

- **Error Correction:**
The error correction facilities amount to an "intelligent" do-what-I-mean [19]. Types of corrections (related to the types of errors described above) include reordering actions, identifying missing activities, supplying a plan to satisfy a required state and substituting parameter bindings that satisfy required constraints.

- **Query Support:**
Queries as to either the state of the world or the state of the actions can be handled, since both states are maintained to support planning functions. Each state represents a "database" of information about current status and past history.

- **Cooperative Automation:**
The automation itself is achieved by planning. Cooperation is accomplished by requesting programmer decisions on such issues as what parameter bindings to use in operators, when to terminate iterated activities, or how to select among alternative operators.

1.4 Achieving Intelligent Assistance

The architecture we have described is an ambitious one. While planning appears to be an appropriate framework for reasoning about a process, the key is being able to capture and utilize all the relevant process knowledge. If too little knowledge is captured, the intelligent assistant will not be able to deliver substantive support, or the support will be rigid and ultimately too constrictive to be useful. The challenge arises because the programming process is at least as complex as any domain previously tackled for a planning application. And certain aspects, such as the inherent "trial and error" nature of programming and the fact that the intelligent assistant is not intended to be fully autonomous, are novel.

In the remainder of this paper, we discuss two problems in capturing appropriate levels of software process knowledge. (In exploring *how* to represent this knowledge in a planning formalism, we will also be exploring exactly *what* the knowledge is.) Both problems are due to limits in the representational adequacy of existing hierarchical plan formalisms. The first problem concerns the adequacy of operator definition languages. We illustrate the

difficulties of capturing such relevant domain knowledge as how to recover when an operator fails or when to use special case operators, and provide a solution based upon *meta-plans* that dynamically transform plans. The second problem lies in the scope and role of the domain state schema. We show how the schema can be extended to encompass a deep model of programming process knowledge. *Non-monotonic reasoning* is then used to compensate for partial knowledge of the extended state. Finally, we describe the GRAPPLE project status and present some conclusions.

2.0 Representing Software Process Operators

Hierarchical plan systems, based on NOAH [16] and NONLIN [18], have strengths both in their planning algorithms and in the nature of their operator definitions. When describing a complex domain, the hierarchical approach is appealing because activities can be defined at different levels of abstraction, with more or less detail as appropriate. Another strength lies in the modularity of operator libraries: following the principles of information-hiding, certain details can be restricted to a small number of operators, and, in general, operators can be written without knowledge of the other operators in the library. In complex domains, cases arise where appropriate expressive power is lacking in the basic operator formalism; attempts to describe certain operators accurately can jeopardize the library modularity, or fail outright. Consider the following problems.

2.1 Limits on Representational Power

Adding special case operators to a library may require that preconditions or subgoals of existing operators be rewritten. For example, when testing a system that is intended to fix certain bugs, the programmer should run the official testcases associated with those bugs, in addition to those testcases that would otherwise be selected. One solution is to write a separate operator covering all testing needed when bugs are being fixed; its precondition restricts its applicability to systems intended to fix bugs. Now there are two operators for testing that are intended to be mutually exclusive. Therefore, the normal operator for testing must specify in its precondition that it is not applicable to cases where bugs are being fixed[1].

---

[1] One could institute a fixed preference strategy to select the operator with the most specific precondition that can be satisfied. However, in general this is overly restrictive — it would prevent a car buyer from financing his purchase by selling stock to raise funds because taking out a car loan is the most narrowly applicable operator.

In other situations, existing operators must be rewritten in artificial ways. Consider testing a system that is about to be released to a customer; such testing should include running the testcases in the regression test suite[2] (again, in additional to normally selected testcases). The precondition for this special operator concerns the existence of a goal to release the system; while the goal formula is expressible using domain predicates, the fact that a goal with this formula is currently instantiated is not expressible in domain terms. The only recourse is to write separate operators with artificially different goals. Then, operators (like *build*) that have testing subgoals will be affected. Thus, the designer of the operators must produce not only a normal *test* operator and a *test-for-release* operator, but also a normal *build* operator and a *build-for-release* operator, to ensure that the right type of testing is performed in all cases.

Expressing special cases with this brute force approach, already attended with disadvantages, breaks down entirely when multiple special cases affect a single operator; the combinatorics are intolerable from the designer's perspective. Special cases are not guaranteed to be simply additive with respect to the normal case. At worst, separate operators must be provided for all *combinations* of special factors.

In dealing with recovery from operator failure, there are problems both in connecting the right recovery operator with a failure situation, and in simply expressing the recovery strategy itself. Sometimes special operators are used for failure recovery, and only for failure recovery; for example, one of the actions for dealing with a compilation failure due to bugs in the compiler is to report the compiler bug. *Report-tool-bug* can be written as an operator, but how will such an operator get instantiated? Missing are the constructs indicating what goals (and therefore what operators) should be instantiated when a failure occurs. At other times, the recovery strategy may involve executing some normal operator in a special way. If the *build* operator fails because the *system* being built is faulty (as would be the case if the linker detected programming errors), then one recovery strategy is to restart the *build* process using the faulty *system* as the *baseline* from which to edit. Expressing such a strategy requires access to the variable bindings of operator instances; again, this is beyond the scope of domain predicates.

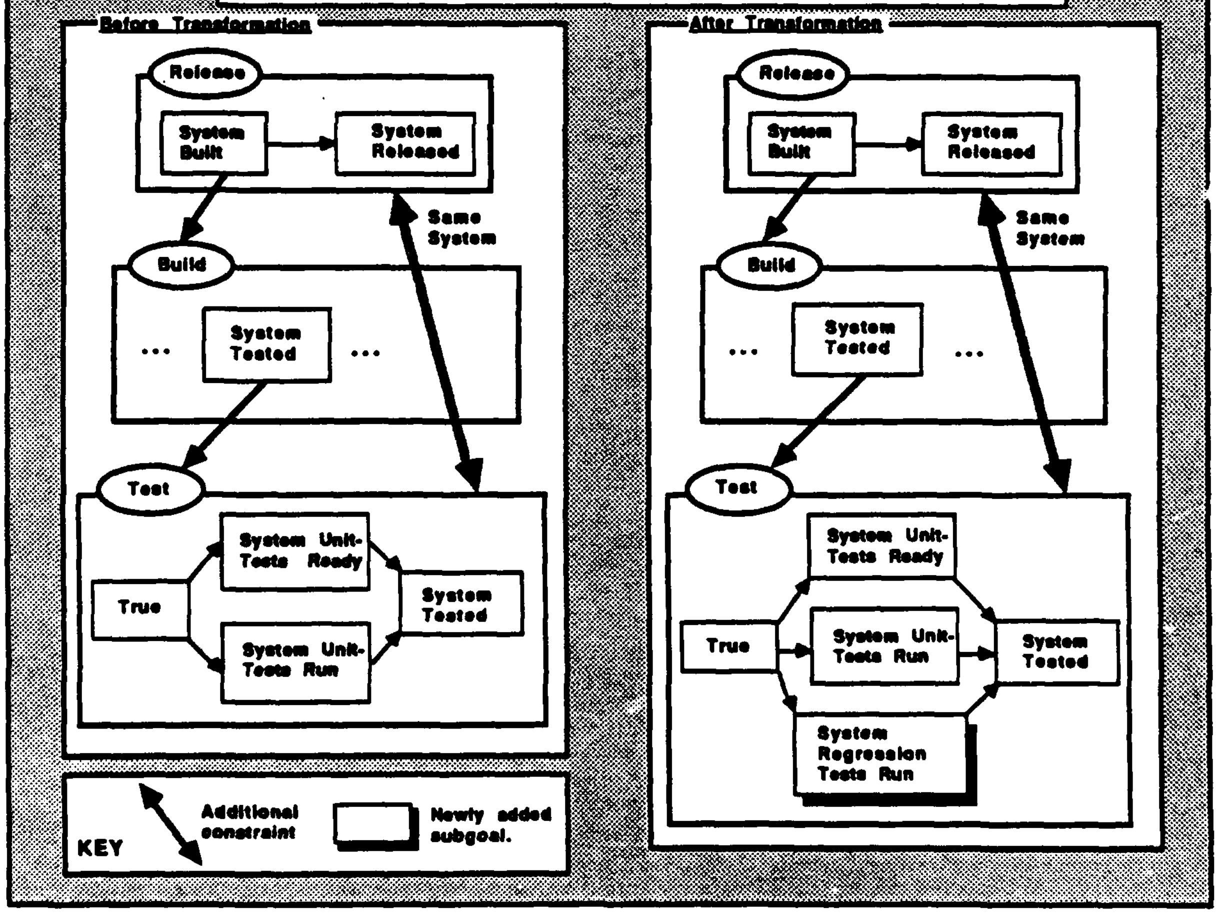
---

[2] Regression testing is performed to ensure that bugs have not been introduced into functions that were previously shown to work correctly.

2.2 Extending Representational Power

The ideal solution would be a single formalism that significantly extends representational power. In the past, limitations have been tackled on a case-by-case basis, introducing special operator-language constructs for each case. McDermott's policies [12] and the error recovery language of SIPE [24] are two examples. What if we abandon the notion of pre-defining *all* operators, and instead applied transformations to *instances* of operators within a plan to create variations in response to special circumstances? Transformations are operations on some world state; in this case, the world state *is* the plan network. Therefore, such transformations can be formalized as meta-operators and synthesized into meta-plans. This approach has the desired generality; it also adds to the role of meta-plans, which have previously been used to implement control strategies [17] and capture domain-independent knowledge [22].

FIGURE 5: Implementing Regression Testing Before Release



2.3 An Example

Consider a transformation that implements the requirement to do regression testing before a release. When expressed precisely, the transformation affects an instance of *test* occurring as part of the expansion of *release*. To be entirely safe, one additional restriction should be given: that the *system* being tested is the same as the *system* being released. This will allow other testing instances to occur in the same expansion (such as running a testcase to help decide what editing changes are needed), while ensuring that regression testing is required on the right one. Expressing this condition requires access to the dynamic correspondence between the variable names used in the two operators. The BEFORE case of Figure 5 shows one situation in which this transformation is applicable.

Assuming the *test* operator of Figure 2, the effect of the transformation is to add an additional subgoal to run the regression test cases. The formula defining the new subgoal is supplied explicitly in the transformation — it need not have appeared previously in the plan network. Only the one operator instance is modified; the basic operator definition for *test* is unchanged. The results of applying this transformation are shown as the AFTER

Figure 6: A Meta-Operator Example

```
METAOPERATOR   regressions-before-release

GOAL           applied-to(regressions-before-release, ?test-op)

PRECOND        instance-of(?test-op, test) AND
               ancestor(?test-op, ?rel-op) AND
               instance-of(?rel-op, release) AND
               same-dynamic-name(system, ?rel-op,
                 system, ?test-op) AND
               NOT applied-to(regressions-before-release, ?test-op)

EFFECTS        NEW state-instance ?regressions
               NEW subgoal-entry ?regr-subgoal
               NEW iteration-spec ?iterate-info
               ADD part-of(?regressions, ?test-op)
               ADD instance-of(?regressions, ?regr-subgoal)
               ADD iterator(?regr-subgoal, ?iterate-info)
               ADD source(?regressions, metaplan)
               ADD role(?regressions, subgoal)
               ADD protection(?regressions, not-protected)
               ADD satisfaction(?regressions, unknown)
               ADD formula(?regr-subgoal,
                     tested-on(?system, ?regr-case) )
               ADD formula(?iterate-info,
                     in-regression-suite(?regr-case) )
               ADD applied-to(regressions-before-release, ?test-op)
```

case in Figure 5. The transformation is expressed formally in GRAPPLE notation in Figure 6.

2.4 Power of Meta-plans

The software domain is particularly rich in opportunities for expressing domain knowledge in transformations. Some additional examples that demonstrate the generality of the transformational approach are as follows:

- In a multi-user system, when the number of users logged-in is below a certain threshold, then commands may be submitted for foreground execution rather than to a background queue. One transformation, applying to all operators utilizing the background queue, can be written in lieu of an additional version of each such operator.

- Recovering from failed operators includes the chore of deleting extraneous files. One transformation could identify certain files created by operators in the expansion of a failed operator and instantiate goals to delete them. This transformation applies one change (instantiate a goal with specific variable bindings) many times (for each selected file).

- The conservative editing style of frequently saving a snapshot of the file being edited also involves eventually deleting the intermediate snapshots. This too is a complex transformation, because the deletions cannot be specified until after the identity of the satisfactory version is known.

- Programmers generally follow a set of rules about how files are allocated to directories. However, in the heat of activity, a file may be created in the "wrong" directory. A transformation could trigger on this and instantiate a goal to move the file to the proper directory. Such transformations reestablish desirable domain states, in the manner of McDermott's primitive policies [12].

- An operation copying one file to another is used expressly for the purpose of preventing conflicts between two subsequent actions, one of which will modify the file and the other of which will use the original form of the file. *Copy* can be written as a normal domain operator, but as in the case of operator failure, some connection still remains to be made between the goal of *copy* and a situation when it is appropriate to instantiate that goal. A transformation can be written to do this; the precondition for the transformation is that an adverse interaction between two planned actions

has been detected. Here a transformation is used to augment the domain-independent forms of untangling operator interactions by defining a domain-specific strategy.

- A well-established strategy for dealing with the compiler having blown up when directed to compile at its highest optimization level is to try again with optimization turned off. If this results in a successful compilation, the programmer will settle for a load module which is only partially optimized, even if performance testing was planned. This transformation should both rephrase the goal to lower the optimization required and instantiate a goal to repeat the performance testing when a fully optimized load module can be produced. This is an instance of McDermott's notion of rephrasing a goal when no plan can be constructed to achieve it.

- If editing a source module consists of cosmetic changes only, then an alternative to recompilation is simply to acquire (and place in the appropriate directory) the object module of the previous version (assuming no include modules were also changed). However, it is bad practice to do this on a final release to a customer. Only by expressing this in a transformation can we ensure that good practice is followed. In this case the goal is rephrased to take advantage of special circumstances.

- Several projects can share the same generic operator library, if each of them implements their project-specific requirements as meta-operators. For example, one project can require that a particular analysis tool be run before a system build is considered finished, without affecting whether other projects also choose to use the same tool in the same way.

In summary, the primary advantage of this method is expressive generality in a single formalism, as compared with a collection of *ad hoc* operator language extensions. Any aspect of an operator definition (such as preconditions, subgoals, constraints, or effects), as well as any aspect of an operator instance (such as bindings of variables or ancestor operator instances) can be accessed or modified; complete technical details appear in [8]. The transformational approach also addresses practical problems in providing a complete library of operators. Because knowledge of exceptions is partitioned from knowledge of normal cases, the two issues can be tackled separately. The process of writing operators is further improved because multiple transformations can apply to a single operator, thereby preventing combinatorial explosion in numbers of operators.

3.0 Representing Software Process State

3.1 Towards a Deeper Model

Whether expressed in operators or meta-operators, the knowledge available to the intelligent assistant is deceptive — it is not as great as it might appear. Achieving substantive support without rigidity requires a still deeper understanding of the domain. Consider a simple example. The *build* operator of Figure 1 contains no constraints on the selection of the baseline from which a new system version is to be developed. In the absence of constraints, the intelligent assistant must forgo opportunities for both error detection and automation; that is, the plan recognizer will not be able to validate a selection made by the programmer, and the planner will not be able to supply a selection automatically. The typical situation could be covered by a constraint requiring that the baseline be the most recent system version. But, use of this constraint results in rigidity because there are times when the constraint is inappropriate. For example, the current customer release is the appropriate baseline when making a bug fix for quick turnaround back to the customer, in order to avoid releasing new code that has not yet been adequately tested.

In reasoning about the programming process, programmers employ a rich model comprised of laws that govern and explain what bugs are, how they occur, why and how systems are built incrementally, and why and how systems consist of modular components. These laws address both surface knowledge directly observable from primitive actions (as captured in the domain state of Figure 3), as well as knowledge which is not directly observable, and may not be readily available to the intelligent assistant. For example, reasoning about baseline choices involves (among other factors) a notion of purpose: is the new system version to be main line development or a prototype? When the building of a new system version starts, its purpose is most likely not identifiable. Acquiring the missing information by the simple expedient of querying the user is not a solution: it would be too invasive, and would have the effect of significantly reducing user productivity.

This issue of partial knowledge is not an *obstacle* to achieving deeper domain understanding. Rather it presents an *opportunity* for furthering the goal of having an assistant that is *independently* intelligent. After all, programmers regularly give advice to one another — even though the advisor has incomplete knowledge of the precise state of the advisee. To capitalize on this opportunity, it will be necessary to do more than

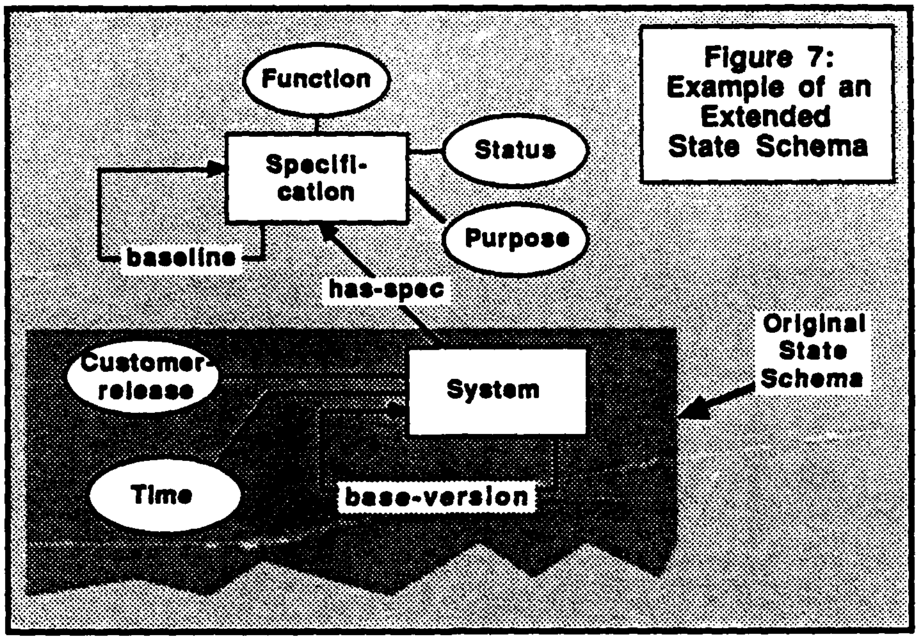


Figure 7: Example of an Extended State Schema

formalize the programming process laws and make the corresponding extensions to the domain state. We must, in addition, select reasoning mechanisms that allow plausible inference in the presence of partial information.

3.2 An Example

As an example, consider a domain state extended by adding a specification entity along with its attributes and relationships, in order to reason about baseline choices in the *build* operator. The extension to the model is shown in Figure 7. There we do not use the term "specification" in any formal sense, but rather as acknowledgment that every system version has a set of criteria it is expected to (but may not) meet; these criteria may not even exist as an on-line textual file, let alone in any more structured representation.

The ideas about a specification that we wish to capture are expressed as axioms[3], given in Figure 8. The principal notion is that each specification (except an initial one) is defined in terms of another specification (using the *baseline* relationship). If two specifications have the same baseline, then only one of them can be the main line of development (*purpose* is

[3] Although not mentioned previously, axioms are also given for the basic state schema of Figure 3. For example, axioms would be used to require that an object module be compiled from exactly one source module and that a bug cannot be reported fixed in a load module that predates the load module the bug was first found in. Such axioms are treated as constraints on values in the world state. If a constraint is violated, then it is an indication that there is an error in plan recognition or planning, indicating the need to backtrack to other choices of operators or variable bindings.

Figure 8: Some Axioms on Extended State

A1:	IF has-spec(sys1,spec1) AND has-spec(sys2,spec1) THEN equal(sys1,sys2)	Each system has a unique specification.
A2:	IF baseline(spec1,spec0) THEN status(spec0,extendable) AND function(spec0,incomplete)	Only a spec which is extendable and of incomplete function can be the baseline of another spec.
A3:	IF baseline(spec1,spec0) AND purpose(spec1,main-dev) THEN purpose(spec0,main-dev)	The purpose of a spec can be main-dev only if its baseline has a purpose of main-dev.
A4:	IF baseline(spec1,spec0) AND purpose(spec0,prototype) THEN purpose(spec1,prototype)	If the purpose of a spec is prototype, then any spec for which it is the baseline must also have a purpose of prototype; and, similarly for a purpose of satisfy-customer.
A5:	IF baseline(spec1,spec0) AND purpose(spec0,satisfy-customer) THEN purpose(spec1,satisfy-customer)	
A6:	IF baseline(spec1,spec0) AND baseline(spec2,spec0) AND NOT equal(spec1,spec2) THEN NOT (purpose(spec1,main-dev) AND purpose(spec2,main-dev))	If there is a fork in development, no more than one branch can have a purpose of main-dev.
A7:	IF baseline(spec1,spec0) AND has-spec(sys0,spec0) AND NOT customer-release(sys0) THEN NOT purpose(spec1,satisfy-customer)	The purpose of a spec cannot be satisfy-customer unless the system of its baseline spec was released to the customer.
A8:	IF NOT purpose(spec1,main-dev) THEN function(spec1,incomplete)	Only a spec whose purpose is main-dev can have complete function.
A9:	IF base-version(sys1,sys0) AND has-spec(sys1,spec1) AND has-spec(sys0,spec0) THEN baseline(spec1,spec0))	If one system is the base-version of another, then the spec of the first must be the baseline of the spec of the second.
A10:	IF purpose(spec1,satisfy-customer) AND baseline(spec1,spec0) AND has-spec(sys1,spec1) AND has-spec(sys0,spec0) AND successor-customer-release(sys2,sys0) THEN equal(sys2,sys1)	If the purpose of a spec is satisfy-customer, then its system must be released to the customer as the successor to the system of its baseline spec.

main-dev); only the main line of development can lead to achieving the ultimate specification (*function* is *complete* as opposed to *incomplete*). If a specification does not represent a main line of development, then either it is a prototype in the sense of throw-away code (*purpose* is *prototype*) or it is an attempt to make "minor" improvements in the current customer release (*purpose* is *satisfy-customer*).[4] Eventually lines of development that are not the main line will die out (*status* is *deadend* as opposed to *extendable*).

With the introduction of the specification, we have necessarily entered the realm of three-valued logic (in particular, the strong system due to Kleene, described in [20]). The third truth value, *unknown*, is used to represent cases of incomplete information. This is exactly as expected: while we know that every system version has a unique specification (axiom A1), we may not know any of the attributes of that specification nor which other specification (if any) is its baseline.

---

[4] For purposes of this example, we assume that there are no other reasons for forking development. We further assume that joins do not occur.

The axioms of Figure 8 are the direct expression of the laws that must be satisfied in a valid software process. However, they can be viewed from another perspective — as a set of rules for deducing attributes and relationships of specifications. Unfortunately, from this perspective, the rule set is deficient: there are cases where the timing of deductions is too late, and cases of incomplete and non-deterministic rules. Axiom A2 allows the deduction that a specification is extendable, at the time it is selected as the baseline for another specification. However, it is preferable to decide if a specification is extendable *before* it is selected as a baseline, precisely in order to validate the selection; thus, axiom A2 (the only axiom dealing with status) fires too late. While there is a rule (axiom A4) for forward propagation of a purpose of prototyping, there is no rule for determining when the first instance of prototyping has occurred. Finally, some rules, such as A6, are non-deterministic, ruling out some choices without ruling in a single remaining choice.

With such a rule set, the intelligent assistant can only record implications of programming actions; in fact, with this particular rule set, contradictions cannot be identified at all (but in general, this will not be true). The ability to form independent judgements in a timely fashion is lacking. This is hardly satisfactory! In order to achieve more aggressive behavior, the intelligent assistant must make default assumptions about the attributes of specifications when there is supporting evidence for such defaults. These assumptions may have to be retracted at a later date when new evidence contradicts the default.

This is exactly the kind of plausible inference which is possible with non-monotonic reasoning, in particular the default logic of Reiter [15]. In Figure 9 we give some example default rules that augment the (monotonic) axioms to remedy the problems described above. A rule of the form *A WITH CONSISTENT B YIELDS C* [5] means that if *A* is true and *B* is consistent with all other known facts and axioms, then *C* may be assumed to be true. For example, the rule D3 captures the notion that any fork in development meant purely to improve upon the current customer release is likely to deadend after one step; this default will rule out the choice of this specification as a baseline for another specification when used in conjunction with axiom A2. The rules D1 and D3 implement reasoning of the form "X is more likely than Y", while rule D2 implements reasoning of the form "if X holds, then Z is evidence for Y".

---

[5] We have departed from the standard format for default rules because it breaks down when *A*, *B*, and *C* are lengthy expressions. The standard format is: $\frac{A\ :MB}{C}$.

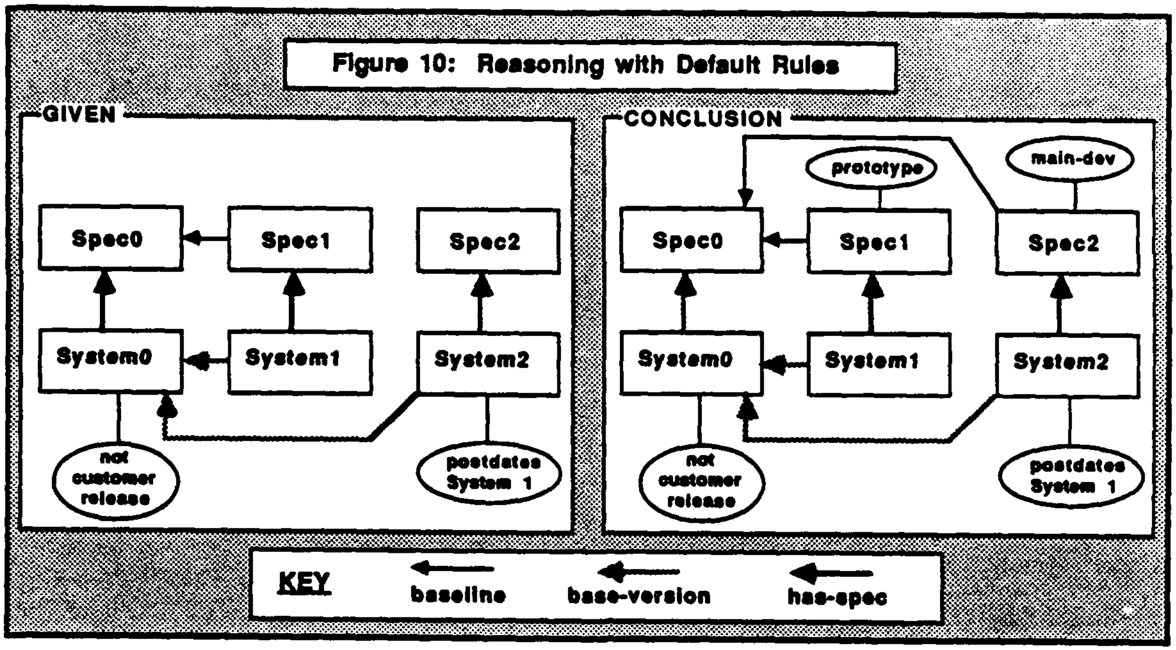
Figure 9: Default Rules on Extended State

D1:	baseline(spec1,spec0) AND baseline(spec2,spec0) AND NOT equal(spec1,spec2) AND has-spec(sys0,spec0) AND has-spec(sys1,spec1) AND has-spec(sys2,spec2) AND NOT customer-release(sys0) AND postdates(sys2,sys1) WITH CONSISTENT purpose(spec2,main-dev) AND purpose(spec1,prototype) YIELDS *purpose(spec2,main-dev) AND purpose(spec1,prototype)*	If there is a fork in development, starting from a system which was not customer released, then assume that the first (chronological) fork is a prototype and the second is main development. (Generally, prototyping precedes production development; *a purpose of satisfying the customer* is already excluded by A7.)
D2:	baseline(spec1,spec0) AND baseline(spec2,spec0) AND NOT equal(spec1,spec2) AND has-spec(sys0,spec0) AND AND has-spec(sys1,spec1) AND has-spec(sys2,spec2) AND customer-release(sys0) AND postdates(sys2,sys1) AND lm-for(sys0,lm0) AND first-found-in(bug1,lm0) AND value-to-customer(bug1,critical) WITH CONSISTENT *purpose(spec2,satisfy-customer)* YIELDS purpose(spec2,satisfy-customer)	If there is a fork in development, starting from a system which was customer released, and there is a bug reported against that system which has high importance to the user, then assume that the second fork has a purpose *of satisfying the customer, remaining* agnostic about the purpose of the first. (The existence of such a bug is "evidence" to support interpreting the purpose to be satisfy-customer.)
D3:	purpose(spec1,satisfy-customer) WITH CONSISTENT status(spec1,deadend) YIELDS status(spec1,deadend)	A spec whose purpose is satisfy-customer is unlikely to be the baseline for another spec.

Figure 10 shows an example of reasoning with the rules and axioms to reach a CONCLUSION from a GIVEN situation. In the CONCLUSION, the fact that spec2 has spec0 as its baseline is due to axiom A9; the facts about the purposes of spec2 and spec1 are due to rule D1; and, the fact that spec0 has a purpose of main-dev is due to A3 (after the application of D1). If the GIVEN situation were to include the fact that the purpose of spec0 is main-dev, then the reasoning would be unchanged, except that A3 would no longer be invoked. If the GIVEN situation were to include the fact that the purpose of spec0 is prototype, then rule D1 would be blocked (it is not consistent that spec2 have a purpose of main-dev when spec0 has a purpose of prototype, by axiom A3); but in this case, no default rule is needed because A4 decrees that the purposes of both spec2 and spec1 must be prototype.

3.3 Comments

In this simple example, we have shown how programming process laws are captured in axioms applying to an extended domain state, and how default rules compensate for the fact that the extended state may be incomplete. The reasoning made possible by this example is not limited to selecting/validating baseline choices. After a baseline choice is made, we have, either by monotonic or non-monotonic means, gathered quite a bit of information about a specification. This information can be used to confirm the choices of test cases and



even to make assumptions about the outcomes of tests, which in turn may influence the type of specification appropriate for the next system version. The synergy is greater when the laws and state model are extended further. Knowledge can be added about the computation defined by a load module and the path through that computation taken by a particular test case. This knowledge can be used to make assumptions about the persistence of bugs through system versions, the blocking of one bug by another, and evidence that bugs have been fixed or new bugs found.

The explicit expression of programming process laws allows *reasoning from first principles*. That is, no set of rules specifically addresses legal choices of baselines as an independent issue. Rather, there are rules that relate this choice to a deeper model involving specifications, and additional rules that allow reasoning within this deeper model. The nature of the default rules determines the degree of certainty in this reasoning; with a suitable set of default rules, the assistant will occasionally draw faulty (but correctable) conclusions, as a result of reasoning independently from the programmer with incomplete information.

The process laws are not *universal* to all software processes. Because they capture constraints to be satisfied if a particular sequence of activities is a valid way to carry out part of a process, the laws are an integral part of the definition of a process. However, differences between two processes are more likely to be accommodated by changes in the operator library than by changes in the process laws. Two very similar processes might

share the same set of laws, with the differences reflected exclusively in the operator/meta-operator library; two somewhat similar processes would share many but not all laws; and, two less similar processes would share only a few laws.

4.0 GRAPPLE Project Status

We are building a testbed for experimentation with the various components of the GRAPPLE intelligent assistant architecture. An initial version of the plan recognizer has been implemented; it uses a plan formalism [7] that we have engineered to meet the demands of complex domains. GRAPPLE is not tied to a particular software environment; rather, it accepts command streams transcribed from actual terminal sessions or fabricated for experimental purposes. The implementation runs on Texas Instruments Explorer™ workstations, and is written in Common Lisp and Knowledge Craft™ (chosen for its facilities for context management, object schema, and integrated Prolog features). Efforts are underway to enlarge the library of software process operators and to implement support for meta-operators. Our current research focus is on the role of the deep model and reasoning from first principles.

5.0 Conclusions

We have described a plan-based approach to supporting the programmer carrying out the process of programming. The types of support that can be provided include generation of agendas and summaries of process status, detection and correction of a wide range of process errors, and cooperative automation of process activities. The use of planning technology has multiple advantages: ability to provide both active and passive support, separation of domain-dependent knowledge in operator libraries from domain-independent knowledge in the planning algorithms, and techniques for handling destructive interactions among sub-plans. However, the success of a plan-based approach is dependent on capturing knowledge of a complex process in a planning formalism. We have identified two issues in representational adequacy of traditional planning formalisms, and have shown that two techniques (meta-plans and non-monotonic reasoning) can be used to significantly extend representational power.

Knowledge Craft™ is a trademark of Carnegie Group Incorporated. Explorer™ is a trademark of Texas Instruments Incorporated.

At the same time, we have explored the breadth and depth of knowledge that programmers have of the programming process. Many examples of this knowledge have been given, covering such questions as what programming goals are, how individual tools are used, when special cases arise, how to recover from different types of failures, and what *first principles* knowledge is relevant. In showing that a planning formalism is capable of expressing and utilizing such knowledge, we have demonstrated that a plan-based approach is very well-suited to supporting the programming process.

6.0 References

[1] Broverman, C.A., and W.B. Croft, "A Knowledge-based Approach to Data Management for Intelligent User Interfaces", *Proceedings of Conference for Very Large Databases*, 1985.

[2] Broverman, C.A., K.E. Huff, and V.R. Lesser, "The Role of Plan Recognition in Intelligent Interface Design", *Proceedings of Conference on Systems, Man and Cybernetics*, IEEE, 1986, pp. 863-868.

[3] Carver, N., V.R. Lesser and D. McCue, "Focusing in Plan Recognition", *Proceedings of AAAI*, 1984, pp. 42-48.

[4] Chen, P.P., "The Entity-relationship Model: Toward A Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, no. 1, March 1976, pp. 9-36.

[5] Croft, W.B., L. Lefkowitz, V.R. Lesser and K.E. Huff, "POISE: An Intelligent Interface for Profession-based Systems", *Conference on Artificial Intelligence*, Oakland, Michigan, 1983.

[6] Croft, W.B., and L.S. Lefkowitz, "Task Support in an Office System", *ACM Transactions on Office Information Systems*, vol. 2, 1984, pp. 197-212.

[7] Huff, K.E. and V.R. Lesser, "The GRAPPLE Plan Formalism", Technical Report 87-08, Department of Computer and Information Sciences, University of Massachusetts, Amherst, 1987.

[8] Huff, K.E. and V.R. Lesser, "Meta-Plans That Dynamically Transform Plans", Technical Report 87-10, Department of Computer and Information Sciences, University of Massachusetts, Amherst, 1987.

[9] Johnson, W. and E. Soloway, "PROUST: Knowledge-Based Program Understanding", *IEEE Transactions on Software Engineering*, vol. 11, no. 3, March 1985, pp. 267-275.

[10] Kaiser, G.E., and P.H. Feiler, "An Architecture for Intelligent Assistance in Software Development", *Proceedings of the Ninth International Conference on Software Engineering*, IEEE, 1987, pp. 180-188.

[11] Lehman, M.M., "Process Models, Process Programs, Programming Support", *Proceedings of the Ninth International Conference on Software Engineering*, IEEE, 1987, pp.14-16.

[12] McDermott, D., "Planning and Acting", *Cognitive Science*, vol. 2, 1978, pp. 71-109.

[13] Nilsson, N.J., *Principles of Artificial Intelligence*, Tioga Publishing, Palo Alto, California, 1980.

[14] Osterweil, L., "Software Processes are Software Too", *Proceedings of the Ninth International Conference on Software Engineering*, IEEE, 1987, pp.2-13.

[15] Reiter, R., "A Logic for Default Reasoning", *Artificial Intelligence*, vol. 13, 1980, pp. 81-132.

[16] Sacerdoti, E.D., *A Structure for Plans and Behavior*, Elsevier-North Holland, New York, 1977.

[17] Stefik, M., "Planning and Meta-planning", *Artificial Intelligence*, vol. 16, 1981, pp. 141-169.

[18] Tate, A., "Project Planning Using a Hierarchical Non-linear Planner", Dept. of Artificial Intelligence Report 25, Edinburgh University, 1976.

[19] Teitelman, W. and L. Masinter, "The Interlisp Programming Environment", *Computer*, vol. 14, no. 4, April 1981, pp. 25-33.

[20] Turner, R., *Logics for Artificial Intelligence*, Ellis Horwood Limited, Chichester, England, 1984.

[21] Waters, R., "The Programmer's Apprentice: A Session with KBEmacs", IEEE *Transactions on Software Engineering*, vol. 11, no. 11, November 1985, pp. 1296-1320.

[22] Wilensky, R., "Meta-Planning: Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding", *Cognitive Science*, vol. 5, 1981, pp. 197-233.

[23] Wilkins, D.E., "Domain-Independent Planning: Representation and Plan Generation", *Artificial Intelligence*, Vol. 22, 1984, pp. 269-301.

[24] Wilkins, D.E., "Recovering From Execution Errors in SIPE", TN 346, SRI International, January, 1985.

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

Appendix 5-B

Building Tools for Tutorial Discourse [1]

Beverly Woolf
Tom Murray
Dan Suthers

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Klaus Schultz

School of Education University of Massachusetts
Amherst, Massachusetts 01003
September 25, 1987

Abstract

We have designed general techniques for managing discourse in an intelligent tutor. These techniques are being implemented in structures that dynamically reason about a tutor's response to the student and customise its generation of examples to the individual student. The structures are flexible, domain-independent, and designed to be rebuilt, i.e., decision points and machine actions are intended to be modified through a visual editor. In this article, we discuss these formal reasoning structures and describe how they are being applied to improve the response of intelligent tutors for physics education.

1 Reasoning about Tutoring Response

Effective tutoring requires sophisticated and dynamic reasoning about the selection of tutoring strategy. Choices, such as which path to take through the curriculum and which examples to generate, will affect the tutor's response to an individual student. Conversational actions produced by an intelligent tutor and responses from students will change the state of the discourse, and the tutor must decide dynamically how to interpret and act on student actions.

[1]This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York, 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-85-C-0008. This grant supports the Northeast Artificial Intelligence Consortium (NAIC).

Hence a desideratum on the design of an intelligent tutor is that it respond fluidly to the user and that it coordinate its utterances in a more flexible manner than has been required for question/answer or summarization systems.

The techniques we have developed improve a tutor's ability to dynamically reason about discourse and to refine its selection of appropriate remediation activities such as example presentation or question asking.

For instance, we have developed an example generation formalism motivated by the need to provide a rich simulation environment to encourage student activity and exploration in new domains. We want students to manipulate concepts and examples "on many levels, from many angles, and with facility and spontaneity. [Students] must be able to travel freely through the environment, to experiment with its items, shift the level of concern from detail to broad overview and vice versa, and be able to ask questions" [Michner (Rissland), 1978]. In order to provide such rich simulation environments, we have developed a framework for representing knowledge, i.e., concepts, examples, and questions, presentations, and rules about using concepts, based on the role that the particular knowledge plays in understanding the domain in general and the theory behind the domain in particular.

In our systems, a partial ordering is imposed on information such as examples or presentations and a way is defined for the tutor to intelligently pass through the space to present this information to the student (Figure 1). In this way we support students in testing their intuitions about the domain. For instance, we can present incrementally more complex or more simple examples based upon student or tutor request. The epistemology we have developed is not complete nor exhaustive, yet we believe it begins to provide knowledge about those elements in a domain that are used when experts do their work and when they explain their knowledge.

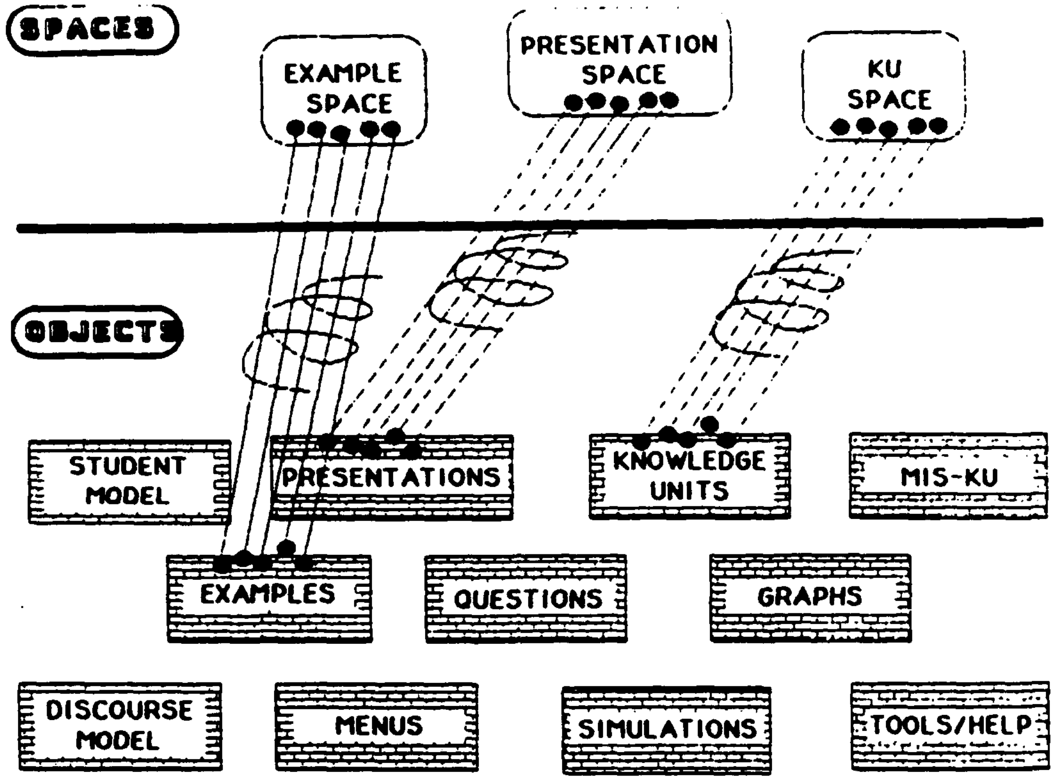


Figure 1: Spaces Defined as a Partial Ordering on Objects

In the past, networks and production rule formalisms have been used to identify admissible instructional strategies and examples (e.g., Cerri & Breuker, 1981; Clancey, 1982). However, such formalisms were often domain-dependent and restricted to a narrow set of didactic responses. The process models we have developed are domain independent, ideally enabling a tutor to transition from one domain to another. The mechanisms allow the tutor to move between situations dynamically, to track discourse contingencies, and to consider discourse alternatives. The knowledge bases and control structures we have built are modular, object-oriented, and extendable; they are designed to be rebuilt.

Fundamental to our perspective is the view that tutoring conversation and the presentation of examples are motivated by general rules (principles) of tutoring and discourse. Such rules are incompletely understood; they are the focus of attention for many researchers, including ourselves and others [Murray et al., submitted AIED Journal; Clement, 1983; Grosz & Sidner, 1985; Reichman, 1985]. Cognitive results from such studies are being used to inform the process models we build. They also serve as a basis of our theory of tutorial discourse.

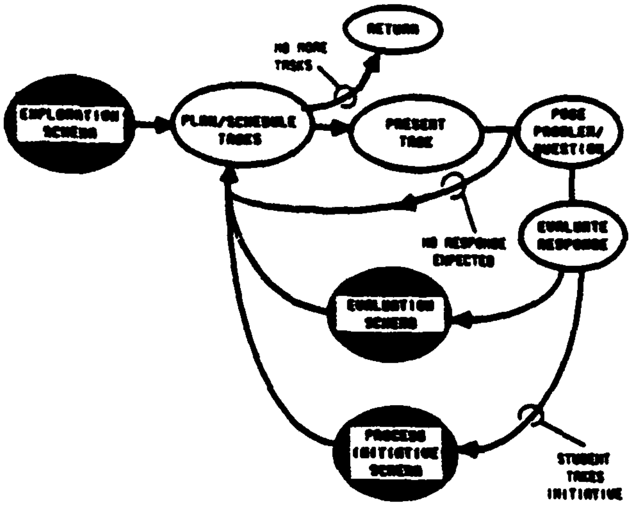
In the next section we describe the discourse management mechanisn we have developed to dynamically guide a tutor's choice of response. In the following section we describe the example generation system that tailors examples to the need of the individual student.

2 Discourse Management

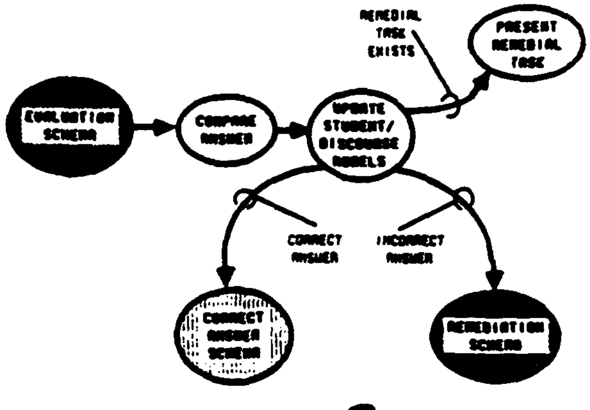
We have built a discourse management mechanism to facilitate context-dependent interpretations of machine responses [Woolf & Murray, 1987]. This discourse manager uses a taxonomy of frequently observed discourse sequences to provide default responses for a machine tutor. It also uses state variables to make choices between alternative discourse sequences. The architecture employs *schema*, or collections of discourse activities and tutoring responses, as shown in Figure 2 to direct discourse transitions. These schema are derived from empirical research into tutoring, including studies of teaching and learning [Brown et al., 1986; Littman et al., 1986], misconception research [Clement, 1982, 1983; Stevens et al, 1982], identification of felicity laws in teaching [van Lehn, 1983], and general rules of discourse structure [Grosz & Sidner, 1985].

We call the space of possible discourse sequences a *Tutoring ACtion Transition Network (TACTN)*, [2] [McDonald et al., 1986]. The machine response is generated by traversal through the formal structure expressed by arcs and nodes. Arcs are defined as predicate sets, which track the state of the conversation, and nodes provide actions for the tutor. The outer loop of the discourse manager first assesses the situation indicated by the arcs, resolving any conflicts between multiply satisfied predicate sets, and then directs the system's other components (e.g., the underlying domain expert component, the language component, or the student model) to carry out the action indicated by the node.

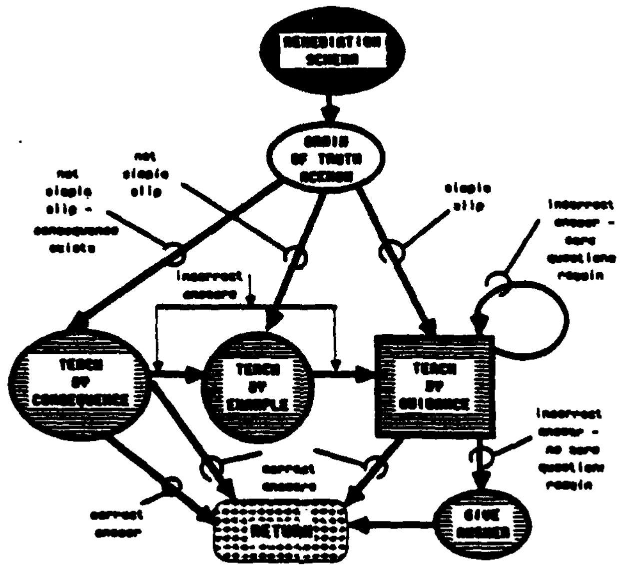
[2] Rhymes with ACT-IN



A
Exploration Schema.



B
Evaluation Schema.



C
Remediation Schema.

Figure 2: Tutoring Discourse Schemata

Discourse control consists of passage through the arcs and nodes of the schema, with the number and type of schema depending on context. For example, if the student's answer is correct (Figure 2A) new schema, notably Evaluation and Correct Answer schema (Figure 1B), will be activated. However, if the student's answer is incorrect, up to six schema might be traversed in sequence, including possibly three schema activated by the Remediation Schema (Figure 2B) as shown in Figure 2C, Teach by Consequence, Teach by Example, and Teach by Guidance. The exact number of schema depends on the tutor's assessment of student error, i.e., whether it was *a simple slip, not a simple slip,* or *not a simple slip—consequence exists.*

In the next section we provide a brief example of how this discourse planning framework is being implemented in a series of physics tutors and in the following section describe the tutoring structure in more detail.

2.1 The Statics Tutor

We are building science tutors as part of the Exploring Systems Earth (ESE) consortium.[3] The science tutors provide interactive simulations whose aim is to encourage students to work with "elements" of physics, such as mass, acceleration, and force. The goal is to aid students in developing problem-solving skills, knowledge about physics concepts, and intuitions to facilitate learning about knowledge and skills.

In these tutors, students explore simulations, such as the one shown in Figure 3, called the crane boom problem. In this example, students are asked to identify forces and torques on a crane boom and wall such that the boom will remain in equilibrium, i.e.,there will be no vertical

[3] ESE is a group of universities working together to develop intelligent science tutors. The schools include the University of Massachusetts, San Francisco State University, and University of Hawaii.

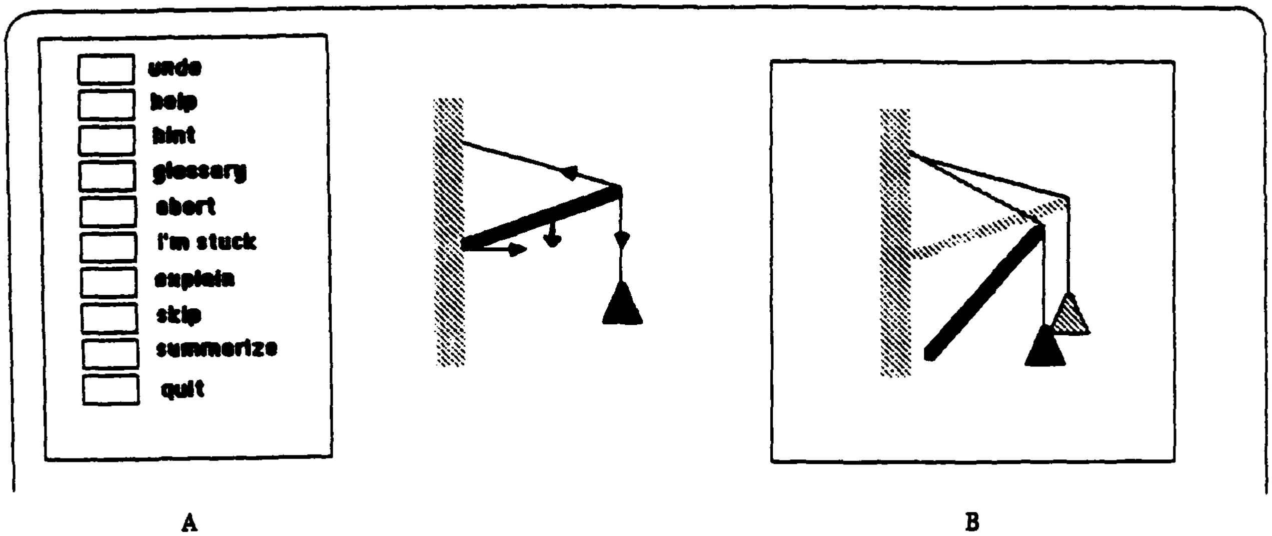


Figure 3: Simulation Presented to the Student: Statics

or horizontal movement. Students must draw appropriate force vectors to solve the problem through vector diagrams or equations.

The Exploration Schema in Figure 2A provides the main control loop for the tutor's interactions about this and other physics problems. It schedules tasks, which might be items from a top-level curriculum list,.or sub-goals generated by earlier tasks. Associated with each task is information about how to present the problem situation, what question(s) to ask, which knowledge is assumed of the student, etc.

For example, if a student had provided the force vectors indicated by heavy vectors in Figure 3a, the solution would be incorrect because a force vector is missing (located at the wall and pointing upwards). The tutor compares the student's response with the correct one and activates the Remediation Schema in Figure 2C. The consequence of the student's actions are indicated in Figure 3b and the conversation produced by the tutor is shown in Figure 4.

situation: – incorrect answer
action: [grain of truth acknowledgment]

T: The beam does experience forces in each place you have shown and in the directions you have shown.

situation: – not simple slip
and consequence exists
action: [teach by consequence]

T: Here is what the beam would look like if it experienced the two forces you indicated (beam is shown pulling away from the wall).
T: Is this OK or would you like to reconsider your answer?
S: It's OK.
situation: – incorrect answer
action: [teach by example]
T: Consider a situation in which a roller skate wheel is attached to the left end of the beam. What would happen to the beam then?
S: It would fall because it would roll down the wall.

situation: – correct answer
action: [compare answers]

T: Compare the two situations–with and without the roller. Do you want to reconsider either of your answers in light of this comparison?
S: No.
situation: – incorrect answer
action: [teach by guidance]
T: You are missing a force somewhere on the beam. Do you want another hint?

Figure 4: Conversation Between Tutor and Student

2.2 Representational Power

TACTN structures have the look of conventional transition graphs of "nodes" and "arcs" as one would find in ATNs [Woods, 1970], yet they are defined in terms of "actions" and "situations" [McDonald et al., 1986] as one would find in a production system. The motivation to modify the ATN architecture was based on the observation that ATNs were designed for natural language <u>parsing</u> and thus are non-deterministic. [4]

Non-determinism has no counterpart in natural language <u>generation</u>, which requires a planning rather than a parsing formalism [McDonald et al., 1986]. Any action in a TACTN can be taken deterministically. For instance, any transition, Teach by Consequence, Teach by Example, or Teach by Guidance, might be taken locally without the need to wait for a global interpretation. Discourse generation requires choosing between alternative actions, not alternative interpretations, placing it in the realm of a planner, not a parser. Since we were not using all the capacity provided by ATNs, we decided to modify the architecture.

Elements from both production systems and network formalisms have been incorporated into our new architecture. Situations, located on arcs, are associated with actions, located on nodes, in the manner of a production system, and discourse history is encoded in arcs and nodes, in the manner of a network system. Every situation/arc implicitly includes as one of its constituent predicates the action(s)/node(s) from which it came. Thus, if an arc originated in a particular action, there is a tacit predicate included in the arc's history that the previous node's action

[4] Nodes in the original ATN represented accepted definitions for incoming tokens while arcs represented tests made on those incoming tokens. Non-determinism was motivated by uncertainty, or the need to wait for an accumulated global interpretation before the system could be confident about the local interpretation of each token being scanned.

must have just taken place. The single notational framework has the flexibility of a production system and the record-keeping and sequencing ability of a network system by virtue of its context.

A pure production system used for discourse manager cannot retain and use a large amount of contextual knowledge to handle complex shifts in dynamic conversation. For example, GUIDON [Clancey, 1982] was a tutor based on a set of situation action rules that were driven by chaining backwards from goals. It provided flexibility to respond dynamically to changing discourse circumstances. However, as there was no provision for sequencing situation action chains except by using ad hoc state variables, GUIDON retained very little contextual knowledge. In TACTNs, the act of chaining to arcs from actions provides just such a sequencing mechanism. A problem arises if multiple paths lead to a node, in which case the tutor does not know which path was taken to move to the node. Thus, we rely on the TACTN author to permit this only when the histories of the arcs, in terms of discourse situation and student model predicates, are equivalent.

Arcs Define Situations Arcs in the discourse structure are defined by a set of predicates that track the student and discourse from the perspective of the system. Arcs correspond to discourse situations. For example, in Figure 2C the arc *simple slip* is a compound predicate that is true under two conditions: (1) the current topic is factual and the student has had medium success with it in the past, or (2) the topic is conceptual and the student has had high success with it. In a sense, situations are abstractions over the state of the system or student knowledge, expressing generalized conditions such as *student takes initiative* or *student is confused*.

The definition of arcs as a structural and logical combination of predicates in a boolean formula

provides a powerful tool for knowledge engineers. Modifying predicates, and therefore arcs, allows fine-grain changes on individual predicates to impact greatly on the system's reasoning ability and to result in consequential changes to the tutor's discourse activity.

Within this structure, several arcs might define nearly equivalent situations as in the case when two arcs share one or more predicates. At such time, a conflict between arcs will be resolved through global or local (associated with specific nodes) conflict resolution strategies.[5] One solution is to order the arcs in the set according to their specificity and to execute the first triggered arc. In this way, the most specific subsuming situation will be preferred over other situations with which the arc shares predicates. However, we prefer to evaluate all the arcs, since incomparable situations (i.e.,situations whose sets of predicates are disjoint) are likely. Additionally, incomparable arcs may indicate situations that the tutor should respond in parallel, rather than in a way that excludes situations that "lose" against other situations.

For example, suppose that students ask many questions after giving a wrong answer, and suppose that they also ask several seemingly random questions. One tutoring convention says that answering students' questions should take priority; another says that random questions should be discouraged. In such a case a non-conventional resolution mechanism must be used to resolve the conflict.

Nodes Define Actions Nodes correspond to actions available to the system; they define alternative conversations and tutoring strategies. Nodes differ in the actions they perform and in the manner they present tasks to students. Shifting actions to the nodes, as we did for TACTNs, instead of leaving them on the arcs, as was done for ATNs, facilitates the notation

[5] Conflict resolution in this sense is analogous to what happens in a production system when the left-hand sides of more than one rule are satisfied.

of expanding abstract actions into more concrete substeps. Abstract actions are nodes that are to be expanded and refined one or more times before taking on a form that can be executed. For example, the node Evaluation Schema (Figure 2a) is an abstract node whose expansion led to a second abstract node called Remediation Schema (Figure 2b). On the other hand, the node Present Task (Figure 2a) is not an abstract node, it represents an immediately executable action. Action expansion is an activity of the discourse manager. This notion of abstract planning borrows principally from Sacerdoti [1974] and Stefik [1981].

2.3 Evaluation of the Discourse Management Mechanism

In building this discourse mechanism and modifying its basic ATN architecture, we have increased the functionality of our earlier domain-dependent formalism [Woolf & McDonald, 1984]. The goal was to allow a tutor to remain flexible while cooperatively engaged in conversation and to continually adjust the discourse to real-time changes engendered in the system either by the tutor or the user. This mechanism allows the tutor to recognize the possibility of multiple discourse paths arising asynchronously depending on current context.

In addition to this discourse mechanism, we have built an example generation tool that enables the tutor to tailor examples to a particular student. The tool is discussed after a motivating example and exploration of alternative teaching strategies.

3 The Thermodynamics Tutor

As part of the set of interactive and monitored simulations built by the Exploring Systems Earth, ESE consortium (Section 2.1), we are implementing a thermodynamics tutor to improve a

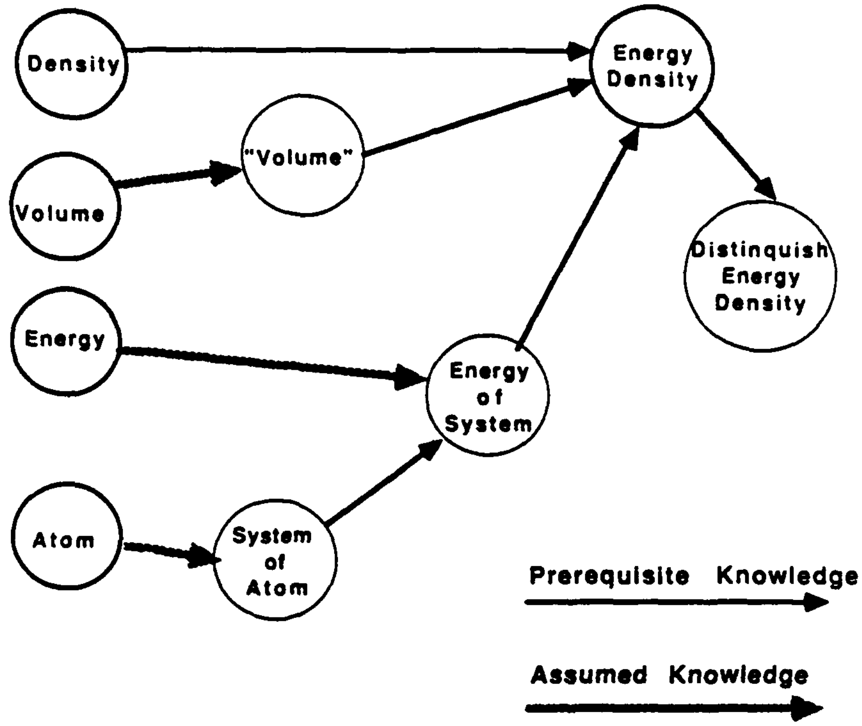


Figure 5: Topics in Thermodynamics

student's intuition about energy, energy density, entropy and equilibrium. Like the statics tutor, this tutor monitors and advises students and provides examples, analogies, or explanations based on student actions, questions, or responses.

In this tutor the goal is to teach the sub-net of topics shown in Figure 5 as a precursor to discussing the second law of thermodynamics.[6] The tutor presents these concepts at the atomic level [Atkins, 1982] and provides a rich environment through which the principles of equilibrium, entropy, and thermal diffusion can be observed and tested.

The student interacts with a simulation that depicts collections of atoms transfering heat to one another through random collision (Figure 6). The student can create areas of "high" energy atoms, indicated by dark squares, and can monitor the flow of energy of the atoms through the regions. As the system moves toward equilibrium, areas of atoms can be monitored and

[6]The second law states that when all the components of a process are taken into consideration, entropy of a system either remains constant or increases.

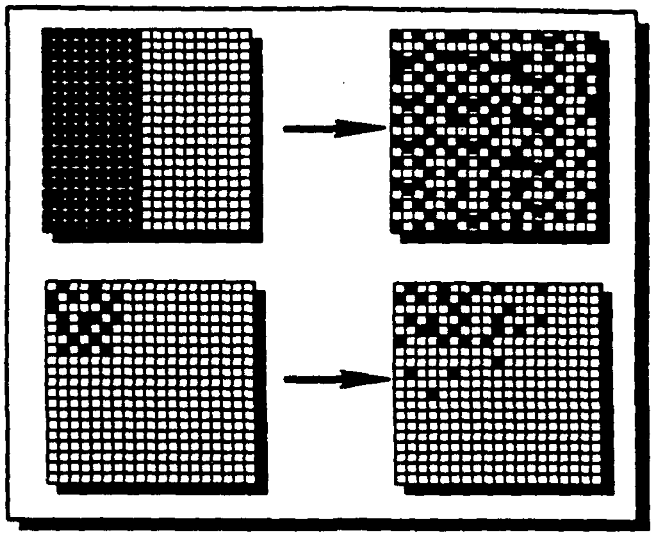


Figure 6: Simulation Presented to the Student: Thermodynamics

analyzed. In this way, several systems can be constructed, each with specific areas of high energy and associated observation regions. Concepts such as temperature, energy density, and thermal equilibrium for each observation region can be plotted against each other and against time. Thermodynamic principles can be observed visually through action rather than statically through formula; heat transference can be observed through random collision and entropy as a function of initial system organization.

At any time the student might modify the number of collisions per unit time, which would correspond to the temperature of the system, and the shape of the observation regions. Changes in these parameters will cause dependent changes in the system.

All student activities, including questions, responses, and requests, are used by the tutor to formulate its next teaching goal and activity. Each student activity is used to reason whether to show an extreme example, or a near-miss one, or to give an analogy, or to ask a question. We are now studying student misconceptions and common errors in thermodynamics and statistics

in order to refine the tutor's response.

4 Example Selection Strategies

To illustrate the importance of example selection for tutoring, we discuss two selection strategies in this section. Currently, only the second strategy, incremental generalization, has been implemented in the thermodynamics tutor described above.

Bridging analogies [Murray et al., submitted] is used when a student has already exhibited misunderstanding of a concept, determined by presenting a relatively difficult case of a concept. Given evidence of a misunderstanding the tutor introduces carefully chosen examples in an effort to teach the concept. First the tutor finds an anchor, or simple case that the student seems to comprehend. Then it presents examples that repeatedly "split the difference" between the most difficult example of the concept the student *understands*, and the simplest example of the concept that the student *misunderstands*. The tutor explicitly asks the student to compare and contrast these two examples, thus engendering a sense of cognitive dissonance until the student sees the similarity in the two examples and changes his mind about the misconceived concept.

For instance, assume the student's intuitions are wrong about whether a table pushes up on books placed on top of it. In this case a convenient anchor, for which the student's intuitions are probably correct, is to ask whether one's hand pushes up on the books when they are placed on top of the hand. In the knowledge space, the relevant examples are organized in an example space according to their attributes, and presumably the anchor and the original problem are "distant" from each other along some feature dimension(s) in this space. The bridging strategy is to establish the student's understanding of the anchor, then find an example which is part way "between" the anchor and the original problem in the example space: this is called the bridge.

The tutor first establishes the analogy between the anchor and the bridge, then establishes the analogy between the bridge and the original problem, which brings the understanding over to the original. If there is difficulty in transferring understanding from the anchor to the bridge, or from the bridge to the original, the entire process may be applied recursively. Various control strategies for recursive bridging are possible, depending on where the bridges are relative to the endpoints of the network.

Incremental generalization is a second selection strategy that differs from the bridging analogy strategy because it works "forward" from a simple understood topic and attempts to generalize the student's knowledge to include qualitative variants of the original example in the same concept. In contrast, the bridging analogy is oriented towards understanding a particular goal example.

Consider an example from the thermodynamics tutor described above. While teaching energy density, the tutor might first present a "start-up example" [Michner (Rissland), 1978] that shows two medium sized regions of atoms with random distributions of energy and asks which has the greater energy density. Assuming the student's answer was correct, the tutor might test the extent of the student's understanding of the concept by varying feature dimensions along sub-ranges that are *irrelevant* to energy density (Figure 7). Thus, the tutor would be generalizing the student's understanding of the concept. For instance, the tutor might present patterned energy distributions to show that energy density is independent of pattern (Figures 7A, 7B, 7C, and 7D). It might also display a system of smaller or larger sized regions that have the same energy density (and thus less or more total energy) as the randomly distributed medium system, to show that energy density is normalized by volume (area in the simulation) (Figures 7E and 7F). Within the feature dimension that is being varied, the strategy is to present feature values that are "closer" to those of the start-up example first, working gradually to the extreme

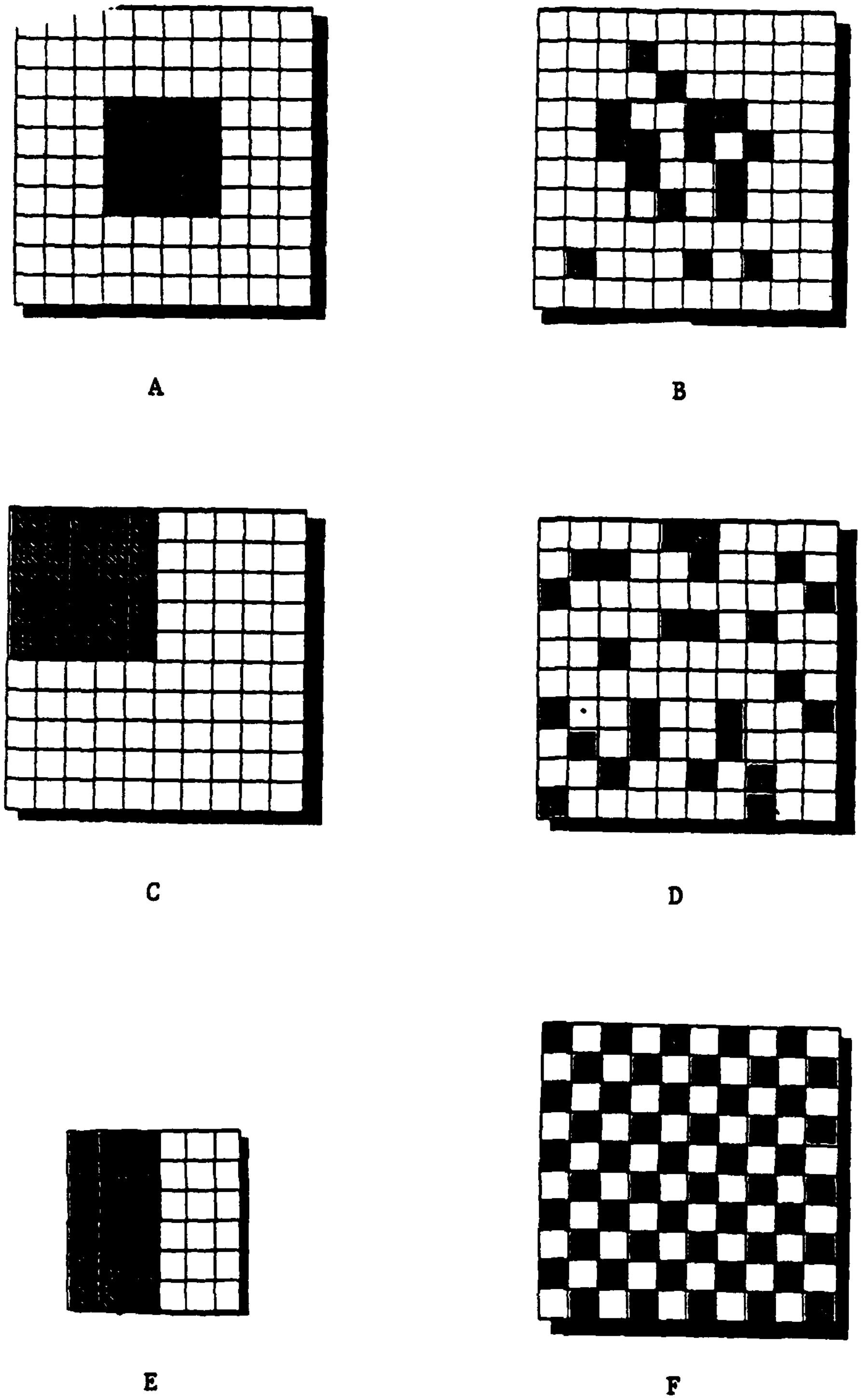


Figure 7: Irrelevant Dimensions of Energy Density

values, thus allowing the student to incrementally generalize the concept. For instance, after generalizing energy density to a regular but uniform (non-random) distribution of energy, the tutor might compare uniform density in one system to another in which all the energy is in one corner of the cube.

Incremental generalization has been implemented in the thermodynamics tutor, as discussed in Section 3 above. The tutor has the potential of exploring all "paths" along feature dimensions from the start-up to their extreme values. A variety of "search" strategies for incremental generalization are possible, as the algorithm leaves open the choice of which feature dimension to vary at any given time. Making these choices will be predicated on the state of a qualitatively rich student model.

5 Example Generation Tool

The second tool for managing tutorial discourse is an example generation mechanism that implements the incremental generalization strategy described above. This mechanism enables a tutor to retrieve and modify examples, tailoring them to the idiosyncrasies of the student and the curriculum. Again, this is a general and domain-independent mechanism that can be built into other tutoring systems and applied to other domains. The tool enables the tutor to make two kinds of control decisions: high level choices that determine which strategy to use in selecting examples and low level choices that determine which example to select next. Both control decisions are based on reasoning about the student's knowledge, the discourse history, and the curriculum.

Decisions at the high level begin or terminate a teaching control strategy; a terminated strategy is replaced with a more appropriate strategy if the student's response pattern has changed. For

instance, if a student shows evidence of misunderstanding an unfamiliar situation, the tutor might replace the current strategy with the bridging analogy strategy to bridge the conceptual gap from the complex situation to simpler and known ones. A library of such strategies will ultimately be available to the tutor along with reasons why it should move from one strategy to another.

Decisions at the low level choose which example to present next. For instance, if the student has made many errors, the tutor might present an example that differs only slightly or perhaps only along a single dimension, from the prior example. On the other hand, another situation might require presentation of a more complex (or more rich or more simple) example. The specification of an example may include questions and problems presented to the student. We intend to generalize the example generation mechanism described below to attempt to handle all tutor-initiated interactions with the student.

Our goal has been to allow the teaching expert to define example selection strategies through general and qualitative, not quantitative rules. We want the machine to respond to directives such as "When the student is in this discourse situation, present more complex examples." This is to be contrasted with situation specific rules such as: "If the student answers question 5 correctly, then present example 32." The problem with the latter kind of specification is that it is responsive only to local information, i.e., which problem was presented and how the student responded. It ignores situation abstractions derived from a sequence of interactions. It also forces the author into a tedious level of detail.

5.1 Traversal of Explicit Example Spaces

Our initial approach to the problem of providing flexible selection and presentation of examples was to place examples in lattice-like spaces designed for the purpose and to traverse these spaces with algorithms representing different example selection strategies (see Figure 1). We encountered some problems in this approach.

The example spaces were defined in terms of the attributes of examples which are relevant to tutoring thermodynamics. For example, when tutoring the concept of energy density, the size, energy pattern, total energy, and energy density of the systems presented are relevant considerations. These feature dimensions define an example space when their values are ordered. Since the pedagogical relevance of feature dimensions differs between topics, a different example space was constructed for each topic. Tutoring strategies were then expressed as algorithms for moving through these spaces. For example, incremental generalization was an algorithm initialized with a pointer to the startup example, which moved this pointer along one feature dimension at a time, in the direction of increasingly "complex" values, the movement of the pointer being controlled by measurements of the student's progress. Bridging analogies was to be implemented in a manner similar to that described in Murray et al. [submitted].

We found that in constructing the example spaces, we had to take into account the tutoring strategy that was to use them. This reduced the strategy-independent utility of explicit example spaces. An undesirable result was that control information was partially encoded in the structure of the space, and partially in the algorithm for traversing it. A second reason for abandoning this approach was the difficulty of taking into account considerations other than the example selection strategy encoded in the traversal algorithm. Decisions which impacted on example selection had been made prior to invocation of the traversal algorithm. Thus what

was intended to be an independent example generation module would have had to look back at other data structures outside itself. A third problem was that the system was limited to using the examples described in the example spaces. To utilize generated examples would have required giving it the ability to insert new examples into the example spaces automatically. If it could do this, it would know enough about the pedagogical utility of examples to do without explicitly represented spaces.

5.2 Example Selection Specialists

The preceding approach did not take into account the variety of considerations that impact on choosing the next example at any given point in the tutor-student interaction. This suggested a more direct representation of these considerations, as recommendations or *requests* produced by a collection of knowledge-source-like entities called *example selection specialists*. [7] These entities or specialists each make recommendations that are dealt with all at once, allowing them to be prioritized and handled with more facility than if their effects were heterogeneously produced. A mechanism for generation of new examples is provided, and these examples may be added to the example base without difficulty.

Example selection specialists are provided by the expert tutor. Each specialist tests the states of the student and discourse models, producing requests as needed to deal with the consideration the specialist was written to handle. For example, one specialist requests that the next example be relevant to the current curriculum topic and another implements the incremental generalization strategy by proposing that the next example have a slightly more extreme value along a given feature dimension than the previous example. A third watches for the appearance

[7] In our current approach, all sources of control knowledge for example selection (the low-level decisions mentioned previously) are represented in this uniform manner.

of a "mis-ku" (representing a misconception) in the student model, and if one is seen, proposes that the current agenda be suspended while remedial tutorials are given. A fourth may try to avoid boredom on the part of the student by requesting that certain salient features of the examples be varied.

A request consists of a request expression, the name of the requesting specialist, and the strength of the request. The expression consists of specified bindings of values to feature dimensions, optionally combined with connectives. Currently, AND, OR, NOT, MEMBER, and SATISFICE (to be described) are implemented. These expressions direct the actions of the retriever and modifier.

5.3 Strategic Phases and Priority

Requests may conflict and the relative importance of the considerations embodied in the different specialists may change as a function of the current situation. Because of this, conflict resolution between requests must be sensitive to the current tutoring strategy. The current implementation uses *strategic phases*, similar to those used in a planner for a medical expert system [Cohen et al., 1987]. The condition of a strategic phase is a function of state variables in the global student and discourse models. Only one strategic phase is active at a time. Among other things, the strategic phase specifies a prioritization of the example generation specialists. Thus the example generation specialists become "terms" in a meta-level control language which decides on the relative importance of these specialists for the situation recognized by the condition of the phase.

IF-NEEDED: function of one argument, an example, which computes the value of the example on this feature dimension.

VALUE-DEPENDS-ON: List of feature dimensions which cannot be modified without also modifying the value on this feature dimension. These are the dimensions referenced by IF-NEEDED in its computation.

TO-MODIFY: Function of two arguments, an example and desired values for this feature dimension, which either modifies the example and returns the value used, or determines that it cannot modify the example (for domain dependent reasons) and returns NIL.

MODIFY-CHANGES: List of feature dimensions which TO-MODIFY changes the value of.

Figure 8: Definition Slots for Feature Dimensions

5.4 Feature Dimensions and Modification

Modification of retrieved examples requires consideration of the interactions between multiple feature dimensions. It may not be possible to modify a given feature without also modifying others. For example, energy-density depends directly on total-energy and the area of the system. Thus, modification is more than just setting a feature dimension slot to a new value. We need to know how to carry out the modification, and what other features are modified.

To provide this information, feature dimensions are defined using the slots shown in Figure 8. The *if-needed* method allows us to compute the value on a feature dimension as a function of

other dimensions of the same example. The *to-modify* method embodies the specific knowledge of how to modify examples, including modification of other feature dimensions besides the one owning the method, as required. The *value-depends-on* and *modify-changes* lists are used for goal protection in the modifier, to be discussed. Note that the feature dimension being defined is always on these two lists.

We have analyzed the interdependencies of the thermodynamics feature dimensions, and have identified those which are *primary*, in that they cannot be computed from other features, and those which are *derived*, in that they can. Derived feature dimensions always have the if-needed and value-depends-on slots defined; and as many feature dimensions as possible have the to-modify and modify-changes slots defined. The values of derived features are always retrieved using the if-needed methods. This permits us to modify examples by modifying only the primary features using the to-modify methods, with the propagation of dependencies to derived features occurring automatically. The advantage is that the modifier code need not contain any domain dependent knowledge about the feature dimensions or their interdependencies.

5.5 Example Retrieval and Modification

The example generator operates within the context of a tutor that maintains the student and discourse models. This includes updating the current topic and the current strategy before a call to the example generator. These higher level control decisions affect the example generator in that the example selection specialists are sensitive to the dynamic models, and the current strategy is used to prioritize their requests.

Example generation begins by allowing the specialists to post their requests, which are then prioritized as specified by the current strategy, but taking into account the strength of the

requests. This results in a list of requests sorted by priority.

Retrieval is done using the SATISFICE command, which when given a list of requests directs the Retriever to return the set of examples which satisfies as many of the requests as possible in the order given, without letting the retrieved set go empty (requests that are not met by any example are ignored). The Retriever need not be concerned with goal protection, since the lower priority requests are allowed to operate only in the set of examples that already satisfy the higher priority ones. One of the (equivalent) retrieved examples is then chosen as the current example.

Before modification, the requests are normalized into a form in which each request is a disjunction of values for one feature dimension in order to simplify the goal protection algorithm and to modify methods. The job of the Modifier is to take the current example, the normalized list of requests, and a record of which of these requests were satisfied, and attempt to satisfy those requests which were not satisfied by retrieval, where it can be done without violating requests of higher priority. This is done by iterating over the requests in priority order, keeping a *protected-features* list, which consists of the union of the value-depends-on slots of the features whose values satisfy requests which have already been checked. The modifier allows the to-modify method of the feature dimension referenced by the current request to run only if its modify-changes slot does not intersect with the current protected-features.

Note that the Modifier is not concerned with how to change values on feature dimensions, or with the fact that dependencies between dimensions require other dimensions be changed in parallel. Its job is simply to determine whether it is OK to run the to-modify method for the feature dimension referenced by a given request, and to record the results.

Modification allows us to tailor examples to the particulars of the current situation, filling out

an example base of "seed" examples provided by the human expert as required to meet the needs of an individual student. If the example was modified, the new example is recorded in the example base. This saves processing if similar situations are encountered in the future. Then the example is returned, to be interpreted by the user interface.

5.6 Status and Evaluation

The retrieval and modification tool described above has been completed in Common Lisp, along with prototype versions of the remainder of the tutoring system in which it must operate. Full evaluation of how well this approach supports the authoring of a tutoring system awaits more complete implementation of the latter system. As of this writing the choice of what topic to tutor next is made by fixed Lisp code that chooses a goal topic, such as energy density, and initializes a topic stack with its prerequisites by visiting them in breadth first order. This will be replaced by a more sensitive controller based on TACTNs. We are examining alternate approaches for implementing a "diagnosis" component for construction of the student model.

The mechanism we have developed for retrieval and modification of examples is flexible and extensible. As a first pass on implementing a robust example generation system, many features require further elaboration. However, the language we have defined is flexible and expressive; it separates representation from control and yet each example is built in terms useful for control. The language is modular ana object oriented, allowing for either examples or selection strategies to be added or changed without rewriting the representation or the control structure.

We are not yet certain about how to most profitably use the identified selection strategies. Bridging is intended for domains in which anchors familiar to the student exist, and where the student is already familiar with the feature dimensions involved (though not their proper

values). Our intuitions are that incremental generalization will prove more appropriate for domains that are new to the student. Neither strategy so far has dealt with how to teach the *negative* instances which are close to the "edge" of the concept. However, this may be a natural extension of how incremental generalization searches the example space; examples could be incrementally modified along the relevant feature dimensions to show how the values of each feature affect the concept being taught.

6 Conclusions

A major research goal of this project has been development of mechanisms for selection and application of discourse responses in tutoring. We have suggested that control of tutoring is isomorphic to control of movement through a space of possible machine responses. In keeping with that model, we have built several knowledge representations and control structures to test how such an approach impacts upon development of human-like tutorial discourse.

Although we still think of example selection and discourse decisions as movement through a space of objects, we have lately rejected an explicit representation of example spaces for implementation. While there are virtues in approaching tutoring as traversal through space, currently our tutor traverses a "virtual" space of tutoring alternatives.

Thus, though we have not settled on a final implementation technique, we have addressed the following issues in development of discourse tools:

1. Representation: Actions of tutorial discourse are represented as nodes and predicates as arcs in a network system. Examples are represented as objects in a partially ordered space. Both representations express possible tutoring responses and both act as media

through which the tutor can move.

2. Control: Rules control movement through the concept network, discourse network, and example space. We are interested in evaluating the relative merit of these strategies and in possibly moving our strategies from the status of algorithms to the level of a unified theory of tutoring.

3. Cognitive Processes: Both representation and control have been developed based on inferred expectations about tutor and student. A human tutor appears to follow implicit rules of discourse and example presentation and a student appears to respond consistently in terms of understood knowledge, explicit errors, and implicit misconceptions. We have begun to represent these expectations as state variable predicates within student model (i.e., student_knows_topic or student_is_unsure). Much work needs to be done to clarify the underlying cognitive process. For instance we need to include more sophisticated components into the model and to recognize how state variables can be interpretable through observable student behavior and how they can be used to prescribe tutorial control.

Ultimately we intend to make the discourse framework and example retrieval systems accessible to human teachers who will modify the knowledge and control structures through visual editors. These visual editors will allow a teacher to use screen figures, similar to those in Figure 2 to reconfigure the machine's response. This is consistent with our long term goal of reducing the excessive time needed for building intelligent tutors by providing structures that can be easily refined and rebuilt as new systems are tested. Both systems are designed to allow a wider circle of authors, e.g., psychologists, teachers, curriculum developers, etc. to participate in the process of implementing intelligent tutors.

7 References

Ashley K., Knowing What to Ask Next and Why: Asking Pertinents Questions Using Cases and Hypotheticals, *Proceedings of the Cognitive Science Society*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

Ashley K., and Rissland, E., Explaining and Arguing with Examples, *Proceedings of the National Association of Artificial Intelligence*, Morgan Kaufmann, Inc., Los Altos, CA, 1984.

Atkins, T., *The Second Law*, Freedman, San Francisco, CA, 1982

Brown, D., Clement, J., and Murray, T., Tutoring Specifications for a Computer Program Which Uses Analogies to Teach Mechanics, *Cognitive Processes Research Group Working Paper*, Department of Physics, University of Massachusetts, Amherst, MA, April 1986.

Cerri, S.A., and Breuker, J., A Rather Intelligent Language Teacher, in Hart (Ed.), *Studies in Language Learning*, University of Illinois, Urbana, IL, Vol. 3, # 1, pp. 182-192, 1981.

Clancey, W., Tutoring rules for Guiding a Case Method Dialogue, in D. Sleeman and J. S. Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, Cambridge, MA, 1982.

Clement, J., Student's Preconceptions in Introductory Mechanics, *American Journal of Physics*, 50 (1), January 1982.

Clement, J., Students' Alternative Conceptions in Mechanics: A Coherent System of Preconceptions? Conference on Student' Misconceptions in Science and Mathematics, Cornell University, Ithica, NY, 1983.

Cohen, P.R., Day, D., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D., and Berman, P., Management of Uncertainty in Medicine, *International Journal of Approximate Reasoning*, 1:103-106, 1987.

Grosz, B., and Sidner, C., The Structures of Discourse Structure, *Proceedings of the National Association of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1985.

Littman, D., Pinto, J., and Soloway, E., An Analysis of Tutorial Reasoning about Programming Bugs, in *Proceedings of the National Association of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1986.

Michner (Rissland), E., Understanding Understanding Mathematics, *Cognitive Science*, Vol. 2, 4, 1978.

McDonald, D., Brooks, J.A., Woolf, B., and Werner, P., Transition Networks for Discourse Management , *COINS Technical Report* # 86-34, 1986.

Murray, T., Brown, T., Clement, J., Schultz, K., and Woolf, B., Using Analogies to Bridge Concepts, submitted to *Artificial Intelligence and Education*.......

Reichman, R., *Making Computers Talk Like You and Me*, MIT Press, 1985.

Rissland (Michner), E., Understanding Understanding Mathematics, *Cognitive Science*, Vol2,

4, 1978.

Sacerdoti, E., Planning in a Hierarchy of Abstraction Spaces, in *Artificial Intelligence*, Vol. 5:2, pp. 115-135, 1974.

Stefik, M., Planning with Constraints, in *Artificial Intelligence*, Vol. 16, pp. 111-140, 1981.

Stevens, A., Collins, A., and Goldin, S., Diagnosing Student's Misconceptions in Causal Models, in D. Sleeman and J.S. Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, Cambridge, MA, 1982.

van Lehn, K., Felicity Conditions for Human Skill Acquisition: Validating an AI Theory, *Report Number CSL-21*, Palo Alto, CA, Xerox Palo Alto Research Center, 1983.

Woods, W., Transition Network Grammars for Natural Language Analysis, *Communications of the ACM*, Vol. 13:10, pp. 591-606, 1970.

Woolf, B., and McDonald, D., Context-dependent Transitions in Tutoring Discourse, *Proceedings of the National Association of Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1984.

Woolf, B., and Murray. T., A Framework for Representing Tutorial Discourse *Proceedings of the International Joint Conference on Artificial Intelligence*, Milan, Italy 1987.

Appendix 5-C

Evidence-Based Plan Recognition

Norman Carver[1]
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

November 1987

[1]This work was supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

Contents

Chapter 1

Introduction

Problems which require the application of artificial intelligence techniques are distinguished by their reliance on incomplete and uncertain knowledge. Plan recognition, the interpretation of data in terms of instances of particular plans, is just such a problem. Combinatorial considerations generally make it impossible to enumerate all of the possible alternative interpretations of the data. Of those interpretations which are considered, evaluation of the relative likelihood of the alternatives is complex and uncertain. In many domains the data may itself be incomplete and/or incorrect.

Previous approaches to plan recognition fail to address many of the issues necessary to produce practical systems. In particular, they provide very limited methods for reasoning about control decisions and for evaluating the appropriateness of alternative interpretations. Because of these concerns, we developed a focus-of-attention scheme for the POISE project [6] which used heuristic knowledge to make the decisions about which hypotheses to pursue. A major advantage of this system was the fact that it provided an explicit representation of the (uncertain) assumptions upon which the control decisions were based. This meant that when interpretation errors were discovered, the system could backtrack, examine earlier decisions, reason about why they were made, and decide how to revise them. While the explicit representation of this control information improved the control process, the approach still suffers from a number of shortcomings. The most important problems include: lack of flexibility for specifying heuristic control knowledge, confusion of belief in an alternative with the decision to pursue the alternative, and little guidance during the revision process.

This paper will develop a new approach to plan recognition which will address the deficiencies of existing systems. The key to this approach is to view plan recognition as a process of *gathering evidence* to *manage uncertainty*. Viewing plan recognition in this way provides a framework within which to apply expert-level heuristic control knowledge and evaluate alternative interpretations. Data is considered as a source of evidence for the plan hypotheses: when data can be interpreted as part of a hypothesis it provides evidence

for that hypothesis. Evidential links are maintained between data and hypotheses the data supports. This provides the system with an explicit representation of the reasons to believe the interpretation hypotheses. The use of an explicit, symbolic representation of evidence is important because it makes it possible to explicitly reason about control decisions. Knowing what evidence supports hypotheses, we can understand the *sources of uncertainty* in the evidence and decide how best to resolve them. When evidence is summarized in numeric degrees of belief, access to this sort of knowledge is lost.

We take a broad view of the class of plan recognition problems. A range of interpretation and situation assessment problems can be viewed as plan recognition problems. The prototypical example is the interpretation of a series of actions as part of some overall task. This ability is relevant to natural language understanding and computerized intelligent assistants. Vehicle monitoring and related situation assessment problems may also be treated as plan recognition problems. Here, the "plans" represent vehicle movements or missions and are composed of characteristic sequences of sensor data rather than "actions." In all of these interpretation problems the goal is to form a higher-level, more abstract view of the data. In other words, to provide an appropriate context within which to understand the data.

Plans specify the hierarchical relations between the data and more abstract views of this data. Although the form and specification of plans varies with the application domain, plans are composed of sets or sequences of subplans. For example, a plan for processing a form is composed of steps for filling the form out and then sending it on to the appropriate office. In the case of vehicle monitoring, a vehicle plan might be composed of subplans which represent sets or sequences of radio and radar emissions which identify the vehicle and its purpose. Plan recognition then, involves the interpretation of sets of subplan instances as (perhaps partial) instances of more abstract plans.

Plan recognition is a complex and uncertain process:

- In general, there are multiple, ambiguous interpretations for each subplan or sequence of subplans.
- Ambiguity is compounded in domains which admit multiple, concurrent plans since the subplans may be interleaved.
- For some applications, interpretation must be done in real-time, relying on preliminary and partial plan data to make "best guesses" about the complete plans.
- Since computational considerations generally preclude constructing all the possible interpretations, there is actually uncertainty whether any of the system's interpretation hypotheses are correct.

- The volume of data may be so massive as to preclude complete examination.
- Data may also be missing, uncertain, and/or incorrect.

As a result of these factors, plan recognition systems must be designed to deal with many uncertainties. We feel that intelligent plan recognition systems must be able to:

- Evaluate the level of belief and uncertainty in alternative interpretations.
- Understand the reasons for beliefs.
- Encode and apply heuristic control knowledge.
- Revise interpretation hypotheses as information accumulates.
- Handle uncertain and incorrect data.
- Integrate data from multiple sources.
- Actively control data accumulation.
- Reflect system goals in control decisions.

Since there will generally be a number of alternative interpretations of any set of data, it is crucial to have some method for evaluating the relative merits of the alternatives. Evaluation has often been accomplished as an implicit part of the control scheme. This is undesirable because it limits potential control strategies to pursuing only the most believed alternatives. Combinatorial and real-time considerations make focus-of-attention strategies crucial. Expert-level heuristic control knowledge can be used if a proper framework is available for encoding it and applying it. However, since control knowledge is fallible and since data may be missing or in error, interpretations should be able to be revised as data is incrementally accumulated.

The final three requirements represent extensions to the normal notion of plan recognition, but have broad applicability nonetheless. In most domains, there are several sources of knowledge which a human expert would use to support his interpretations. Plan recognition systems have typically failed to make use of multiple sources of evidence despite its advantages in dealing with uncertain, incomplete, or incorrect data. For example, in aircraft monitoring applications there would be data from several types of sensors as well as information about terrain and air defenses. Active control could be used to greatly reduce processing effort and system uncertainty when possible. In an aircraft monitoring system some sensors may automatically produce data while others may be controlled by the interpretation system. In an intelligent assistant, the system may prefer to query the

user rather than waiting for additional data to resolve uncertainties. The goals the interpretation process in a domain need not always be the same. An aircraft monitoring system may be trying to protect a sensitive installation or may simply be trying to monitor all air traffic. It may be under time constraints or it may not. The system should adjust its operation to best meet the specific goals–monitoring all aircraft or only the potentially hostile ones, for instance.

We believe that the plan recognition requirements outlined above can be met by viewing plan recognition as a process of *gathering evidence* to *manage uncertainty*. The key characteristics of the approach are:

- Plan, subplan relations are treated as uncertain, evidential relations.
- Evidence and sources of uncertainty are explicitly represented.
- Heuristic control decisions are based on the sources of uncertainty in the hypotheses and the need to manage uncertainty.

Treating plan, subplan relations as evidential relations rather than as absolute goal, subgoal relations means viewing these relations as uncertain inferences. This approach helps to address several of the limitations of existing plan recognition systems. An evidential reasoning system can now be used to provide a representation of the evidence for the alternative hypotheses. This allows their relative likelihoods to be evaluated independent of the control decisions. Reasoning about control decisions can be easily extended to all stages and levels of the interpretation process because the abstraction of any hypothesis is simply an inference. In particular, we need not wait for the construction of top-level plan hypotheses before applying focusing knowledge. Incomplete, uncertain, and incorrect data are naturally accommodated as they simply result in additional sources of uncertainty which can be resolved by gathering sufficient additional evidence. Different types of data can also be easily accommodated because they simply represent different sources of evidence for the interpretations. Revision is a natural part of the accumulation of evidence as conflicting data produces uncertainties which the system can represent and resolve.

By explicit, symbolic representations for evidence, we simply mean that we maintain explicit links between hypotheses and the reasons we believe the hypotheses. Sources of evidence include subplan hypotheses and knowledge such as terrain and weather information. Access to detailed information about the evidence makes it possible for us to make use of an important body of expert-level knowledge about the task: the *sources of uncertainty* in the evidence. In plan recognition, evidence is rarely conclusive. The sources of uncertainty represent the reasons why evidence may fail to support a particular conclusion. For example, acoustic sensor data may fail to support a vehicle because it is actually the result of a sensor malfunction or sensor ghosting. The control component can now reason about

the best course of action for the interpretation system to take because it understands the purpose of its actions: to try to resolve the sources of uncertainty in the hypotheses. An independent, explicit representation of the evidence also makes it possible to represent the relations between the hypotheses. Thus, though direct evidence for a hypothesis may not be available, there may be sources of evidence for related hypotheses–like alternatives.

Active control of data accumulation is possible since the control process explicitly considers the sources of uncertainty in an interpretation and can direct the action of data sources to produce evidence to resolve the uncertainty. Of course, the amount of control the interpretation system can exercise over the evidence it has available will depend on the domain. For example, in vehicle monitoring applications, the operation of a radar sensor can be tuned in order best resolve uncertainty in a particular aircraft ID or location. As system goals change, the importance of different uncertainties changes. When evidential support is summarized numerically it is impossible to consider the context in making control decisions. Maintaining an explicit representation of evidence makes it possible to accommodate varying system goals because beliefs and decisions can be made sensitive to the goal context. The criticality of the uncertainties can be judged in relation to the current state and purpose of the system.

Chapters 2 through 4 motivate this work by examining existing approaches to control, *evidence, and plan recognition.* Chapter 2 is an introduction to control issues as they relate to plan recognition. An overview of existing approaches to representing and using evidence is contained in chapter 3. Chapter 4 presents the plan recognition and related interpretation problems which motivate this research. POISE, the Distributed Vehicle Monitoring Testbed, and several other plan recognition systems are examined. In chapter 5 we present our preliminary view of how evidence and uncertainty should be used in plan recognition systems. Finally, chapter 6 summarizes the goals of this research.

Chapter 2

Control

One of the major problems facing any AI system is the control problem: what should the system do next? An AI system can be viewed as proceeding through a sequence of states as it runs and, in general, there will be several possible actions which could be taken in each of these states. This results in a sequence of choice points for the control component. For example, the control problem for several AI paradigms includes:

Search —which path to pursue.

Plan Recognition —which interpretations to pursue (focus-of-attention).

Production Systems —which satisfied production to "fire" (conflict resolution).

Problem-Solving —which subgoal to work on next.

An important characteristic of AI problem domains is the uncertain and incomplete nature of the knowledge available for problem solving. Thus, control decisions in AI systems are uncertain because the systems will not have complete, correct information which would allow them to choose the right action to take at each decision point. The control component will have to decide the "best" action to take at each decision point based on inexact, heuristic knowledge. Even if more "exact" decision procedures are available for AI problems, the data and knowledge upon which these decisions would be based will be uncertain, incomplete, and/or incorrect: models of the state of the world, models of the state of the problem solving, applicability of operators, etc.

Because of the uncertainty inherent in AI control decisions, problem solving systems must be designed to cope with uncertainty and the resulting incorrect decisions. A number of different approaches to managing this uncertainty have been used in AI systems: using domain-dependent heuristic knowledge to guide the decision process, pursuing multiple alternatives (delaying decisions), backtracking to revise decisions when inconsistencies develop, and opportunistic control with likelihood measures, etc. The appropriate approach depends on the characteristics of the problem-solver and of the application domain.

Regardless of the approach taken to managing uncertainty, as problem solving proceeds, additional knowledge is accumulated which can be used to reduce the uncertainty in the control decisions. Unfortunately, traditional AI systems have suffered from what Doyle has termed the "fatal flaw" of "inaccessibility of control information" [18] due to the implicit nature of most reasoning. This leads to problems which Doyle labels the "inexpressibility of control information" and the "inexplicability of actions and attitudes." Because programs are unable to reason explicitly about why they should or should not take particular actions, it is impossible to encode and maintain sophisticated heuristic control knowledge. Likewise, it is impossible for programs to reason about how to revise their decisions in the face of new evidence since they cannot understand why they did or did not take particular actions.

In the next two sections we will discuss control in AI problem-solvers in terms of two distinct processes: the process of making (the initial) control decisions and the process of revising control decisions. The use of meta-rules and dependency-directed backtracking will be presented as examples of intelligent approaches to control and revision. These techniques have had a strong influence on this work so it is instructive to examine their limitations.

2.1 Control Decisions

Control decisions in AI systems have typically been made in a two stage process. Davis [13] terms these two stages the "retrieval" stage and the "refinement" stage. The retrieval stage determines the actions which may plausibly be taken given the current state of problem-solving. Typically, this stage is implemented using some kind of knowledge indexing scheme appropriate to the characteristics of the problem. A number of different retrieval/indexing paradigms have been used in AI problem-solvers: data-directed, goal-directed, difference-directed, etc. In general, however, such retrieval strategies do not produce a single permissible action at each choice point. That is, indexing schemes alone cannot be used to select the single, correct action to take due to the uncertain and incomplete nature of knowledge in AI systems. Some additional decision procedure must be applied to select the single action to be taken. This is the purpose of the refinement stage–also known as the conflict resolution stage in a production system.

AI systems have often used simple, static refinement procedures. For example, conflict resolution schemes in production systems have selected the "first" rule satisfied, the rule using data most recently added to the database, or the most "specific" rule. A more sophisticated approach to refinement involves the use of numeric ratings or certainty/likelihood factors. In these schemes, ratings or weights are computed for each of the alternatives from the retrieval stage and the most highly rated alternative is chosen. Ratings can be based on various attributes of the relevant alternatives and have included subjective strength

of belief, utility, salience, and a priori probabilities. Numeric approaches provide a more dynamic control than is possible with static approaches since a number of characteristics of the particular relevant data can be considered.

Neither static nor numeric approaches are suitable for use with a truly intelligent control system, however, because they lack explicit knowledge about their decisions. In particular, the intelligence that can be applied to the revision of decisions is severely limited by the implicit nature of the decision process. Static refinement approaches do not explicitly consider any characteristics of the alternatives–their choices are based on fixed, implicit selection criteria. In this situation, it is difficult to see how any revision process could perform better than a blind search. For example, suppose a decision based on the "most recent data" criteria proves to be incorrect. Presumably this decision criteria has been used due to certain assumptions about the nature of the problem solver and/or domain. Since these assumptions are not explicitly considered in the refinement process, they cannot be considered during the revision process. It is impossible now to reason about how to proceed. Should the alternative using the next most recently created data be pursued–is the basis of the decision still sound and applicable? Or, does the failure of the decision indicate that the criteria is invalid? Perhaps it is then appropriate to pursue the alternative based on the oldest data or, perhaps, the age of the data has absolutely no relevance to the correctness of the decision. Without more explicit knowledge about the decision criteria it is impossible to say.

Numeric refinement approaches are not substantially better than static approaches with respect to intelligent control and revision. Their "reasoning" is implicit in their rating calculations and so is unavailable for introspection. Suppose, for example, that ratings are based on the "quality" of the data and the "quality" of the inference rule to which the data is applied. It is impossible to distinguish between an alternative rated highly due to good data and a mediocre inference rule and one which is rated highly due to mediocre data and a good inference rule. All a revision process has to work with is the final numeric rating which condenses the relevant factors–it cannot consider them separately. This makes it impossible to assign blame for failures and take this knowledge into consideration by examining how alternatives are related to the failure. Should inferences based on the same data be tried next because the inference rule is likely the cause of the failure or should this same data be avoided because it was likely the cause of the failure?

Davis [13] and others have advocated viewing the process of making control decisions as a problem solving task in itself. A body of meta-knowledge including heuristic information about the domain and the (object-level) actions would be used to guide the refinement process. Such a refinement process can "reason" about the control decisions by considering a number of different control criteria in the decision process. For a production system, the addition of meta-level knowledge might take the form of a set of meta-rules with

a meta-rule interpreter. The meta-rules would be applied during the conflict-resolution stage of the production system to select a single object-level rule to be fired. For example, a meta-rule from [13] for the domain of stock analysis is:

```
IF the LEI index has been climbing steadily for the past several months
   AND there are rules which mention recession in their premise
THEN it is likely (.7) that these rules will not be useful.
```

While meta-level knowledge embodied in the form of meta-rules allows a system to explicitly consider various characteristics of potential actions, it is clear that many of the control problems we have discussed are still present. In particular, when several meta-rules may be applicable, their advice is typically combined by resorting to numeric ratings. Now, however, we are right back where we started since the object-level factors explicitly considered by the meta-rules are lost in the conversion to numeric ratings. About all that has been gained is a more modular representation for the numeric ratings calculations. In particular, this approach does not provide enough depth or structure to the meta-knowledge and the way it is applied to allow the integration of distinct, but relevant information. Davis [13] recognizes these problems and states that there may be "situations where the rationale behind an argument is as important a factor as its strength." For example, the meta-rule given above limits the use of certain (object-level) stock analysis rules in a particular context. Implicit in this meta-rule is the assumption that a recession is unlikely under the specified conditions and the knowledge that a great deal of processing will be required to recognize this fact. Meta-rules which give conflicting orderings for decisions may be doing so based upon conflicting assumptions about the occurrence of a recession. If the recession assumptions were made explicit, it would be possible to use additional evidence about the likelihood of a recession to resolve the conflicting rankings. Such knowledge could be available either from "external" sources (the user, a backtracking process, or meta-meta-rules) or from other meta-rules which assume that a recession is or is not likely. Even if it were not available, the conflicting rankings could be recognized as alternatives to be resolved as additional evidence is accumulated by the system. This information could then be used to focus and control the system.

2.2 Revising Control Decisions

The uncertain nature of control knowledge means that systems will make errors in their control decisions. When a contradiction occurs or a dead end is reached during problem solving, it is necessary to revise some control decision(s). In general, this requires determining which decisions were responsible for the difficulty, retracting these decisions and their consequences, and making new decisions. However, retraction can be handled in

different ways depending on the characteristics of the domain and the characteristics of the control scheme.

Actual retraction of an action may not be required because the application of an incorrect action does not preclude a successful subsequent search for the solution. Systems that exhibit this characteristic are known as commutative. For example, the application of a "wrong" theorem in a theorem-proving system simply results in the derivation of facts which are useless with respect to the goal of proving the desired fact. The derived facts are still true statements and will not prevent the derivation of the desired result by the application of the "correct" theorems. Nonetheless, a "revision" process is still required to determine which results are useless in order to "prune" the search space and avoid combinatorial explosion in the search, e.g., uncontrolled antecedent deductions in a theorem-proving system. This points up the fact that though answer generation is commutative in such systems, control of problem solving is not due to resource limitations [37]. In other cases, retraction of actions might not be required because multiple paths were pursued by the problem solver, e.g., exhaustive, breadth-first, or beam searches. Again, a revision procedure is required in order to assign blame for the failure and prune the search space in an appropriate manner. These approaches have limited applicability, however. Exhaustive searches are seldom practical for AI problems due to combinatorial explosion in search paths or cost of searching incorrect paths. Partial searches can only avoid retraction if they can guarantee that they cover the correct solution–which can be difficult.

With *chronological backtracking* the state/context of the system is switched to that existing prior to the application of the decision being retracted. This retracts the decision and its consequences. Chronological backtracking is normally implemented so that a contradiction causes the most recent control decision to be retracted. If the temporal order of decisions is not of primary importance, however, the result is a blind, depth-first search for the relevant decision(s). This is exceedingly inefficient because decisions which are not responsible for the inconsistency are unnecessarily withdrawn and reconsidered. In addition to problems with exponential search complexity, much valuable information is discarded in the context switches. This includes information about the inconsistency which led to the retraction and about the paths which have already been explored. In certain domains, the best that can be done is to retract all actions back to the point of the incorrect action–i.e., each action depends on the previous action. Even in these cases, information about the reason for retraction and the paths which had been explored can be useful for determining which paths to explore next and how explore them.

Nonchronological backtracking is one response to the inefficiencies of chronological backtracking. In its purest form, nonchronological backtracking involves examining all decisions, identifying the source of the inconsistency, and choosing an alternative. Thus, only

those decisions which could be responsible for the problem need be withdrawn and reconsidered and only the affected aspects of the state/context need to be retracted. This greatly reduces the search space, although complexity is still exponential in the number of *relevant* decisions.

The best understood implementations of nonchronological backtracking involve a technique known as ***dependency-directed backtracking.*** In order to make use of this technique, dependency records are used to link conclusions with their antecedents. This results in a dependency network which is stored and maintained by a *reason maintenance system* or *RMS* [17,42]. An RMS is a domain independent, syntactic database subsystem for representing propositional deductions and maintaining their consistency. Statement justifications, in the form of dependency links, can be traced back from the inconsistent statements to locate the set of statements upon which the inconsistent statements depend. Certain statements are usually considered to be *premises* or *assumptions.* The RMS is able to change its belief in these statements in order to affect belief in deduced statements and eliminate the inconsistency.

Dependency-directed backtracking is an implementation of the concept of nonchronological backtracking, but the two terms are often confused and used interchangeably [44,52]. This results in nonchronological backtracking appearing to be better understood than it is and dependency-directed backtracking appearing more general than it is. Winston [52] even supplies an "algorithm" for nonchronological backtracking which is similar in style to that provided for chronological backtracking. Unfortunately, unlike the chronological backtracking algorithm, the nonchronological backtracking "algorithm" cannot be implemented. In particular, it does not explain how relevant decisions are to be located, how to go about revising the relevant decisions once located, nor how to proceed with problem solving once the inconsistency is eliminated.

The class of problems to which dependency-directed backtracking can be applied is limited. It is best suited to problems where an explicit constraint network can be constructed such as constraint satisfaction problems and theorem-proving problems. From the point of view of control, the major conceptual failing of dependency-directed backtracking is that it separates control into two disjoint subsystems. The conventional system control component, or problem solver, is here only responsible for the initial control decisions. Revision is handled by the database through the dependency-directed backtracking routines. Dividing the problem solving responsibilities makes it difficult to coordinate control. This leads to a number of problems which will be discussed below.

Because reason maintenance systems only represent *propositional* deduction, instantiation of facts becomes a major control issue. If dependency-directed backtracking is to automatically and independently revise assumptions, the normal control process must effectively pursue all possible alternatives in order to be able to explicitly instantiate them

in the dependency network. Doyle [17] recognizes that this is impossible in domains which involve many alternatives (or domains where alternatives are expensive to compute–see chapter 4), but his proposed solution involves an external process which must somehow be coordinated with the backtracking process. This coordination is difficult to achieve because separating responsibilities for control separates decisions from decision points and decision processes. It is unclear, then, how to re-examine a "decision point" and restart the decision process because the decision point is not explicitly represented in the dependency network in connection with the decision alternatives. This is the reason for what deKleer [16] calls the unouting problem–it is difficult to pursue previously abandoned problem solving paths because the states of the problem solver are not represented in the RMS.

Because dependency-directed backtracking is part of a domain independent subsystem, it is a purely syntactic process, unable to reason about the semantics of a situation when revising decisions. The normal control component of the problem solver must precompute all control information which might be required by the backtracking process and store it within the dependency network. Nonmonotonic dependencies are used to encode sets or sequences of alternative assumptions [17]. These predetermined sequences are then used to revise assumptions without regard to the semantics of the inconsistency. Recording control information in the dependency network means that dependency network dependencies not only record logical deductive relationships, i.e., reasons for believing some facts based on belief in certain other facts, but also control choices. This confuses the role that nonmonotonic justifications play in the nonmonotonic logic notion of default reasoning. Also, the use of little or no intelligence in the revision process severely limits the value of dependency-directed backtracking in real world problem domains. deKleer [16] reports that for qualitative reasoning, RMSs are very inefficient. The reasoner spends most of its processing time backtracking because it must still perform an exhaustive search on the relevant assumptions and revision of the database for each possibility takes a substantial amount of time.

RMS dependencies are typically used to represent only logical deductions, but logical deductions are not the only relations between beliefs and facts which need to be represented. Non-logical inferences, evidential relations, causal connections, etc., will be needed in order to locate relevant assumptions and to reason about their consequences during the revision process. Of course, relations other than logical deduction can be recorded in an RMS by simply including a node to represent the relation among the justifications of a conclusion. However, the semantics of such relations are not understood by the backtracking process and so cannot be used to guide revision. Only one form of non-logical inference, nonmonotonic or default inference [17], can be represented as an integral part of the RMS/dependency-directed backtracking formalism.

Even when explicit dependencies link decisions it is not always obvious which decisions

are "relevant" to eliminating the contradiction. In the planning example in [52], decisions are not independent because of interacting constraints on system resources. When constraints are violated, it is easy to follow dependencies to locate those decisions which are (directly) relevant to the contradiction because of their use of the violated resource. However, these "relevant" decisions are not independent of other decisions with which they share resources. To develop a completely consistent plan may require withdrawing certain of these "irrelevant," but related decisions (a problem which is ignored in the text). How is this to be done without resorting to exhaustive, exponential search?

Finally, it must be noted that the recognition of a "contradiction" or a "dead-end" might involve major problem solving activity of its own [30]. The constraint satisfaction problems to which dependency-directed backtracking has been applied have tended to gloss over the difficulties involved in detecting contradictions and attributing reasons for the contradictions. A measure of developing uncertainty could play an important role in forewarning of these "contradictions."

Chapter 3

Evidential Reasoning

One approach to problem solving in uncertain domains is to apply evidential reasoning techniques. In this chapter we will survey techniques which have been used to represent evidence and belief. The purpose of the chapter is not to examine these approaches in detail, but rather to provide a review of their strengths and weaknesses as they relate to plan recognition. The presentation is divided between those techniques which use numeric representations of evidence and those which use symbolic representations.

Evidential reasoning has typically been applied to AI problems using the parallel certainty inference approach. Inferences are made using two fairly independent processes: conclusions are first derived as if they result from deductive (certain) inferences and then the degree of belief in the conclusion is computed. Computing the degree of belief in the conclusion requires the use of two *combining functions*. *Propagation* combining functions adjust the belief in a conclusion to reflect the belief in the premise and the characteristics of the deduction. *Pooling* combining functions determine the belief in a conclusion which is deduced by multiple independent inferences.

AI systems using evidential reasoning techniques have most commonly involved classification or diagnosis problems. In these problems, evidence is being accumulated to select the most likely hypothesis out of a *fixed set of alternatives*. The fact that all three of the numeric techniques discussed below rely on the existence of a fixed set of alternatives makes it clear why their application has been limited. Plan recognition, for instance, does not fit into the classification framework. Hypotheses are created dynamically as part of the evidence gathering process and can be modified by the very evidence gathered to support the hypotheses. Hypotheses are also frequently interrelated. Because of this, it is questionable whether existing numeric approaches to evidential reasoning are applicable to plan recognition. Symbolic representations of evidence while not inherently limited to diagnosis have primarily been applied in this area and so offer little guidance for the use of symbolic evidence in plan recognition.

3.1 Numeric Evidence

By numeric systems of evidence, we mean those systems which represent their belief, evidence, and/or uncertainty in terms of one or more numbers. AI systems have used a variety of ad hoc numeric rating schemes. However, we focus here on three formal or semi-formal evidential reasoning systems: Bayesian probability, the Dempster-Shafer Theory of Evidence, and the MYCIN certainty-factor model.

Bayes' theorem provides one approach to pooling evidence. In a system based on Bayes' theorem, a single degree of belief would be attached to each hypothesis. These degrees of belief would then be treated as probabilities and Bayes' theorem used to compute the conditional probability of a conclusion given the set of evidence hypotheses. Bayes' theorem has a formal basis in probability theory, but suffers from many deficiencies in relation to its use in an evidential reasoning system: large amounts of data about a priori conditional and joint probabilities are required, but rarely available for domains of interest, the complete set of hypotheses must be known in advance, and these hypotheses must be independent.

The Bayesian approach is unable to distinguish between uncertainty and ignorance because it forces probability to be assigned to singleton sets of the possible conclusions. The Dempster-Shafer theory [25] rectifies this problem by allowing belief to be assigned to any subset of the possible conclusions. Thus if we have evidence which results in a degree of belief x in one conclusion, but no other evidence is available, we can represent our ignorance in the belief in each of the other conclusions by assigning the remaining belief to the subset consisting of these conclusions. This representation also allows us to represent the amount of uncertainty of a hypothesis. Since the evidence not supporting a conclusion need not support the negation of the conclusion, a belief interval is produced which is bounded by the belief in the conclusion and its plausibility–the extent to which the evidence allows one to fail to doubt the conclusion. Despite these advantages, the Dempster-Shafer theory still requires that the set of hypotheses under consideration be mutually exclusive and exhaustive. In addition, the computations required by the Dempster-Shafer theory can become intractable under certain conditions.

One of the major problems with both the Bayesian and Dempster-Shafer approaches is that subjective degrees of belief data used in AI systems does do not represent true probabilities and so these approaches are not applicable. The MYCIN certainty-factor approach [47] was developed as a model of how the kind of non-probabilistic and unformalized reasoning typically carried out by experts could be captured in a computerized reasoning system. In the case of expert knowledge, the data acquired is highly subjective and uncertain and it's impossible to explore the all of the conditional probabilities and interrelationships of the hypotheses. The certainty-factor model recognizes these facts and avoids the problems through use of a simplified, approximate application of Bayes'

theorem.

The intelligence in numeric systems is in the specification of the combining functions. That is, in the process for computing the numbers. Control schemes based on numeric evidence schemes simply select the best rated alternative. As was discussed in section 2.1, a numeric approach to representing evidence means that the control scheme cannot be flexible and dynamic since the reasoning behind the numeric ratings is unavailable. Typically numbers representing expert judgements are a combination of many different factors. For example, in MYCIN, rule qualifications included information about a priori probabilities, causal connections, and utility. This brings up the question of representational adequacy. Numbers do not provide adequate knowledge about situations to allow intelligent control decisions to be made because they only implicitly represent the many different factors relevant to the situations. Doyle [21] has suggested that the inference qualifications represented by the numeric factors be explicitly represented as part of the specification of the inference rule.

3.2 Symbolic Evidence

In response to the deficiencies of numeric representations of evidence, symbolic representations of evidence have been developed. This approach has received far less research attention than have numeric systems. Therefore, the work presented here does not constitute the kind of formalized methods developed for numeric approaches to representing evidence. Instead, it has served to define the complex problems which much be solved in order to use symbolic evidence effectively. The discussion of endorsements summarizes the main arguments for symbolic representations of evidence and the problems which must be solved to use such systems effectively. MUM, a system under development which uses symbolic evidence for the control of medical diagnosis, represents a recent approach to heuristic reasoning about uncertainty.

Numeric degrees of belief implicitly represent summaries of the reasons for believing and disbelieving the conclusions and the inference rules to which they are connected. As we have discussed in sections 2.1 and 3.1, the fact that the numbers hide the reasoning which has produced them severely limits the reasoning that can be done with them. Systems are unable to treat different kinds of evidence differently or to treat evidence differently in different contexts since the only characteristic which is accessible is how much it is believed. Numbers simply do not provide a rich enough representation to support the kind of reasoning that people use to function effectively in the face of uncertain knowledge.

The theory of *endorsements* [9,10] is one response to the limitations of numeric evidence through the use of symbolic representations of evidence. Endorsements are explicit representations of the factors which affect one's certainty in a conclusion. Cohen stresses

the distinction between reasoning *under* uncertainty in numeric systems as opposed to the potential for reasoning *about* uncertainty using a system of endorsements. Since so much more can be known about the evidence, heuristic knowledge can be brought to bear to discount the effect of the uncertainty in a particular context. Reasoning about uncertainty is a knowledge intensive process in which domain specific heuristic knowledge is applied to make the best control decisions given the evidence and the situation. For example, a set of endorsements may be sufficient for one goal, but not for another–in which case the decision could be made to attempt to gather more evidence.

A system which represents evidence symbolically isn't straightforward to develop. If more sophisticated reasoning about evidence is to be done then much more knowledge is required. A system of symbolic evidence doesn't make this any easier–what it does is make it possible to represent and apply such knowledge. Each domain will have a characteristic set of endorsements and a set of methods for reasoning with the endorsements. These methods must include rules for ranking sets of endorsements, rules for combining endorsements (to replace the pooling and propagation combining functions), and rules for resolving and discounting uncertainty. This involves a great deal of information because instead of the uniform, global approaches for dealing with evidence in numeric systems, heuristic reasoning about uncertainty must take into account the characteristics of the context and the particular evidence involved.

MUM (Management of Uncertainty in Medicine) [11] is a medical diagnosis and consultation system which is designed to manage the uncertainty inherent in medical diagnosis. MUM generates workups for chest and abdominal pain. This involves taking histories, asking for physical findings, ordering tests, and prescribing trial therapies. Control decisions are made by reasoning about features of evidence and sources of uncertainty in order to minimize uncertainty or its consequences. The architecture is "based on the idea that managing uncertainty and controlling a complex knowledge system are manifestations of a single task, namely, acquiring evidence and using it to solve problems."

MUM uses a number of different kinds of knowledge. The most basic type of knowledge is *data*, which includes such things as personal and family history and test results. Data must be abstracted through *interpretation functions* to become *evidence*. Interpretation functions are essentially *belief curves* that relate data attributes to evidence or belief in evidence. For example, data about the number of cigarettes smoked per day is abstracted to evidence about the smoking category of the patient: non-smoker, light-smoker, moderate-smoker, or heavy-smoker. In other cases, data is related to belief in a single form of evidence, as duration of chest pain is abstracted to belief in classic-anginal-pain.

Evidence can be characterized by a number of features such as cost to obtain, reliability, and roles. *Roles* represent the relations evidence can play with respect to evaluating belief in hypotheses. MUM recognizes seven roles: confirming, disconfirming, support-

ing, detracting, triggering, and modifying. The roles which relate directly to belief in a hypothesis are realized in the non-numeric degrees of belief used in MUM: confirmed, strongly-supported, supported, unknown, detracted, strongly-detracted, and disconfirmed. Evidence plays a triggering role when it focuses attention on a particular hypothesis. Modifying evidence does not affect belief in a hypothesis so much as it alters the way diagnosis of the hypothesis proceeds. Evidence can play multiple roles with respect to hypotheses. For example, most triggering evidence is also supporting individually or in combination with other evidence. Collections of evidence which occur regularly and play particular roles with respect to hypotheses are grouped into *clusters*. Systems which use representations of belief need to be able to combine evidence and propagate belief. MUM uses *combining functions* which are local to each evidence and disease cluster to accomplish these functions. By doing this instead of using some global, general function, an expert can precisely specify how belief in evidence affects the belief in a cluster.

Strategic knowledge consists of heuristic knowledge for focus-of-attention and actions for gathering pertinent evidence. Strategies are represented as rules and include the following components: conditions for selection of the strategy, focus policies, and planning criteria. Focus policies guide the choice of disease hypotheses to focus on based on plausibility, criticality, or ability to provide alternate/differential explanations for symptoms of the hypotheses. Planning criteria use cost, roles, and diagnosticity (ability to differentiate alternatives) of the potential evidence to control actions to gather evidence.

More recent work on MUM takes advantage of the fact that for medical diagnosis the inference net from data to diagnosis is static and predetermined. As was discussed above, this is not the case for plan recognition. Medical diagnosis is much more a matter of template matching than is plan recognition. This is not, of course, to say that medical diagnosis is any easier than plan recognition since there are still many uncertainties in the domain. It simply suggests that techniques for control in MUM are unlikely to be directly applicable to plan recognition tasks.

Chapter 4

Plan Recognition

We include a broad range of interpretation and situation assessment applications within the class of plan recognition problems we are studying. The classic example of plan recognition is the interpretation of a series of user actions as particular steps in a task instance. This capability is important for natural language understanding systems which must interpret descriptions of user activities or as part of an intelligent assistant such as POISE (see section 4.1). Situation assessment applications such as vehicle monitoring may also be treated as plan recognition problems. Here, for example, the plans describe vehicle movements or missions and the plan "steps" specify characteristic sensor data and other evidence. What is common to these various applications is the goal of producing a higher-level, more abstract view of the data. These interpretations then provide an appropriate context within which to understand the data.

Plan specifications define the hierarchical relations between the data and more abstract views of the data. Although the exact form and specification of plans varies with the application domain, all plans are composed of subplans. These subplans are then further decomposed into subplans. The decomposition continues until the subplans represent available data. For example, an office domain plan for processing a form may be composed of steps for filling out the form and then sending it to the appropriate office for verification. These steps are further decomposed until they correspond to the actions that a user may take using an automated office system. An aircraft monitoring plan might represent a particular kind of mission in terms of vehicle movements and states. The vehicle movements and states would be expressed in terms of of radio and radar emissions necessary to identifying them. The exact form for specifying plans depends on the domain. The subplans of a plan may be explicitly specified in a hierarchical script-like framework or may be more loosely related through goal and subgoal relations depending upon the application. Each plan also has an associated set of parameters. Plan constraints then define the legal subplan instantiations of a given plan instantiation based on various attributes of the subplan parameters. Thus, different forms may be sent to different offices for verification during

processing and different vehicles will have different emissions characteristics.

Plan recognition is the kind of complex, uncertain process which requires the application of AI techniques. The major factors which complicate plan recognition are:

- In general, there are multiple, ambiguous interpretations for each subplan or sequence of subplans.
- Ambiguity is compounded in domains which admit multiple, concurrent plans since the subplans may be interleaved.
- For some applications, interpretation must be done in real-time, relying on preliminary and partial plan data to make "best guesses" about complete plans.
- Since computational considerations generally preclude constructing all the possible interpretations, there is actually uncertainty whether any of the system's interpretation hypotheses are correct.
- The volume of data may be so massive as to preclude complete examination.
- Data may also be missing, uncertain, and/or incorrect.

There is typically insufficient constraint information associated with a plan instantiation to be able to eliminate all but the one correct interpretation from consideration. Of course, the situation improves rapidly if all subplan instances are associated with a single plan since constraint information would accumulate rapidly. However, in many domains multiple, concurrent plan instantiations may occur. For example, users may temporarily suspend a task to start another while waiting for a form to be verified in an intelligent assistant application or multiple vehicles may need to be simultaneously monitored in a vehicle monitoring system. Many applications also require that interpretation be done in real-time–i.e., as the data is being received. An intelligent assistant must try to understand user actions as they are taken if it is to provide maximum assistance and vehicle monitoring is typically required to provide immediate feedback such as in air traffic control. This greatly increases the possib interpretations for a sequence of actions since plan instantiations must be recognized from fragments of the plan and no potential partial plan instantiation may be ruled out as it may be continued later. Becuse these characteristics can lead to a combinatorial explosion of the number of potential ambiguous interpretations for the data, it is often infeasible to construct and evaluate every possible interpretation. Control schemes must instead select and pursue only the "most likely" interpretations. Since these control decisions are uncertain and may be incorrect, there is even the possibility that the correct interpretations are not represented among the constructed hypotheses. Thus, the control

process must not only choose the most likely hypotheses, but also decide if the hypotheses cover the correct answer. In some domains, control must also be exercised over the selection of what data to interpret. For example in vehicle monitoring, many sensors will provide continuous output resulting in huge amounts of data for interpretation. Control of data interpretation is even more critical when the data may be in error. The potential for missing or incorrect data greatly increases the number of potential interpretations since we may be uncertain about ruling out plans just because subplans are missing or because constraints are violated.

In this chapter we will examine the POISE and DVMT applications which have motivated this research as well as several other plan recognition systems. The final section of the chapter discusses the characteristics that we feel plan recognition systems must have to address the deficiencies of existing systems.

4.1 The POISE Intelligent Assistant Project

The POISE project [31] involved the development of an intelligent assistant for users of computerized systems. The project encompasses a number of different components. The purpose of the plan recognition component is the development of a model of user activities based on information supplied by another component which monitors the interactions between the user and the computer. In this section, plan recognition in POISE will be presented along with a discussion of the current control scheme and its deficiencies.

In POISE, possible user tasks are represented as hierarchies of script-like plans. Each plan specifies its substeps using a shuffle grammar to denote the relative temporal ordering of the substeps. Additional constraints specify valid values for the parameters of these substeps. The POISE plan recognition component uses these plan specifications to form an interpretation of user actions. "Primitive" plan instantiations representing the actions are passed to the recognition component by the monitor component. Choice points occur following each newly monitored user action as the system must find an interpretation which covers the latest action. Recognition must be done in real-time in order to provide timely assistance to the user. Thus only partial plan instantiation data will be available and no potential interpretations can be absolutely ruled out due to the possibility of their being continued later. Typically, a user will be engaged in a number of tasks at the same time since most tasks require many steps which take place over a period of time. This means that the plan recognition component must also deal with multiple, concurrent partial plans whose steps are interleaved. Errors made by the user must be considered in the interpretation process. One of the important functions of an intelligent assistant is the detection and correction of user errors. An unexpected action could be due to a user error or it might be due to previous incorrect system interpretations. The recognition

component must assign blame for errors and correct its interpretations when possible. If the error originates with the user, the interface would be notified and a dialog might be carried out to inform the user and correct the error.

Because of these complexities, constraint information contained in the plan specifications is seldom sufficient to fully disambiguate potential action interpretations. Nevertheless, the role of the plan recognition component in modeling user activities for an intelligent assistant makes it crucial that the correct interpretations be rapidly and reliably formed. In order to accomplish this objective, POISE uses heuristic knowledge to focus on the most likely interpretations to be pursued. While the plans contain object-level knowledge about how it is possible to accomplish various tasks, the focusing heuristics contain meta-level knowledge about how people tend to carry out tasks. This heuristic knowledge in effect supplements the knowledge in the plan specifications in order to disambiguate the alternative interpretations. When faced with ambiguous interpretations for the data, the heuristics are used to make assumptions about the most likely interpretations. Of course, since these assumptions are based on heuristic knowledge, the system must include methods for revising the interpretations as additional data is accumulated.

The original focusing algorithm developed for POISE examined all of the possible interpretations for a new action, ordered them, and selected enough of the more likely ones to cover all actions. The heuristic meta-knowledge was implicit in the part of the focusing algorithm which ordered the alternatives. Procedural embedding of the heuristics means that it's not obvious which heuristics have been applied. This caused major problems when actions occurred which were inconsistent with the existing interpretations. This means that the system had incorrectly interpreted some earlier action(s) and needs to backtrack: identify its interpretation error, retract it, and correct it. However, the only information available to the backtracking system was an (ordered) list of preferred interpretations– the *result* of the focusing *process*. There was no information about how this ranking was achieved, nor was there any information about which interpretations were alternatives based on the control assumptions implicit in the focusing algorithm. Thus, there was no way to reason about which interpretations were likely to be in error and the alternatives to pursue instead. Consider for example, the plans I = a, b and J = b, c and the sequence of actions, a, b, c. The system was unable to reason that the interpretations of action b as occurring in plan I or in plan J (Iab vs. Jbc) were alternatives based on an assumption of plans steps being unshared. Thus at this point in the interpretation process, the system selected {Iab, Jbc} as the best interpretations rather than {Ia, Jbc} (which is more correct based on the heuristics). Because it had no explicit record of the interpretation assumptions it had made and the consequences of those assumptions, it would have been necessary for such a system to resort to chronological backtracking in order to reach the correct conclusion.

The current approach to focusing [6] uses meta-level knowledge in the form of heuristic rules similar to [13] (see section 2.1). These heuristic rules result in pairwise orderings of interpretation alternatives which are explicitly recorded and serve as the basis of the interpretation decisions. A list of meta-rules which justify the orderings are also recorded using an RMS. Backtracking occurs when an action cannot be interpreted within the existing task interpretations. Decisions relevant to the error are located and the heuristic reasons for these decisions examined. If a meta-level heuristic rule results in what is deemed an incorrect interpretation decision, then the rule is made inapplicable at the decision point. The RMS then uses the heuristic justifications to make different assumptions about the relative likelihoods of possible interpretations which results in an updated interpretation decision.

While this focusing scheme has a number of advantages, some of the control difficulties discussed in chapter 2 remain. Although the focusing mechanism makes use of an RMS to record and enforce focusing assumptions, the RMS is not used for automatic dependency-directed backtracking. Focusing in POISE is an example of a problem which is not amenable to dependency-directed backtracking, but can be approached through some form of *nonchronological* backtracking (as was discussed in section 2.2). Dependency-directed backtracking cannot be used because of its requirement that all relevant knowledge be instantiated in the dependency network so that inconsistencies can be detected and eliminated. This would essentially require that all possible interpretations be pursued and recorded–including those deemed unlikely. Avoiding this work, though, is exactly the point of focusing because it is prohibitively expensive to pursue all interpretations and because it dilutes the value of the system as an intelligent assistant. Thus, the nonchronological backtracking mechanism is external to the RMS. Relevant assumptions are located by determining which control assumptions resulted in the elimination of task interpretations which could explain the current action and the heuristic focusing rules applied to the revised view of the situation.

The focusing heuristics have been structured in a form like Doyle's *reasoned assumptions* [21]: A UNLESS B ASSUME C. Encoding control knowledge in this way has a number of drawbacks. We must explicitly state when and only when to make an assumption or else inconsistent assumptions may be suggested. Having to precisely specify the exact conditions makes the heuristics complex and difficult to accurately specify. In addition, such a representation is not modular since the addition of new heuristics may require changes to the existing heuristics. Using numeric ratings to resolve conflicting heuristics as has been done for meta-rules (see section 2.1) is not acceptable since it eliminates the kind of explicit representation of control knowledge which is required for an intelligent revision process.

The heuristic control knowledge in POISE is only applied in a limited way to control

the construction of interpretation hypotheses. For each new action, the system constructs all possible interpretations for that action (given the existing interpretation assumptions) in terms of top-level task descriptions before it applies any heuristic focusing knowledge. While this approach seems satisfactory for an intelligent assistant for office automation, it may not work in domains such as software engineering where many tasks are accomplished with few primitive actions. This leads to a large branching factor and hence a very large number of potential alternative interpretations. In this case it may be necessary to apply heuristic knowledge to all plan levels during the construction of interpretation hypotheses. It may even be desirable to limit the abstraction level at which data is interpreted until sufficient data is accumulated to provide some level of certainty.

The current system confuses belief in hypotheses with the control decisions about how to develop the hypotheses. Belief in an interpretation hypothesis is implicit in its membership in the set of currently "in-focus" hypotheses. This often forces the system to prematurely commit to one of a set of alternative interpretations regardless of how tenuous the evidence is. However, when there is a great deal of uncertainty over the proper interpretations it might be better to pursue interpretations other than those that are currently most supported. This would give the system control over using a depth-first vs. a breadth-first approach to pursuing hypotheses. An independent representation of belief in hypotheses would also make it possible to provide more information to a user about the relative level of belief and uncertainty in the alternatives. This same knowledge could be used to guide and limit the revision process. Uncertain assumptions could be examined first and revisions can be limited to those assumptions which are not strongly believed. This is important since it is computationally infeasible to reconsider all interpretation decisions when faced with a contradiction. It may in fact, be possible to recognize incorrect decisions before a "hard" error is caused by evaluating the uncertainty in the system.

4.2 The Distributed Vehicle Monitoring Testbed

The Distributed Vehicle Monitoring Testbed (DVMT) [12] is a research environment for the evaluation of alternative designs for distributed problem solving networks. The vehicle monitoring task involves the generation of maps of vehicle movements through some geographical area. Input data is provided by a set of acoustic sensors distributed over the area to be monitored. Because of the distributed nature of the acoustic sensors, there can be advantages to distributing the computational resources. This requires a problem solving architecture which makes it possible for each node to work with only partial information by communicating with other nodes and by coordinating its problem solving activities with these nodes. The testbed simulates a network of nodes, each of which is a complete, goal-directed blackboard system capable of functioning as a centralized vehicle monitoring

system.

The vehicle monitoring task can be formulated as an interpretation task very similar in character to plan recognition in POISE. The interpretation of acoustic sensor data involves the use of a simple, four-level grammar (plan) representing vehicle tracks in terms of characteristic acoustic sensor data. The data blackboard is divided along these four levels of abstraction. At the lowest level of abstraction, the *signal* level, hypotheses correspond to signal data received from low-level analysis of acoustic sensor data. The *group* level involves collections of harmonically related signals–signals emanating from a common source vehicle. Vehicles are represented as collections of groups associated with particular vehicle types at the *vehicle* level of the blackboard. Finally, the *pattern* level involves collections of vehicles with specific spatial relationships as well as single vehicles. At each level, hypotheses may represent single locations or tracks covering a sequence of locations. The grammar specifies relations between classes of hypotheses from one level to the next as well as constraints such as vehicle velocity and acceleration. The goal of the DVMT is to create pattern-level track hypotheses representing the vehicle movements being monitored by the acoustic sensors.

Vehicle monitoring is an inherently uncertain task. The number of vehicles being monitored is unknown. Constraints in the track grammar are fairly weak. Sensors can fail to detect signals, malfunction and "detect" non-existent signals, or incorrectly determine the location and frequency of signals. These factors result in large numbers of alternative interpretations for a set of signal data–ambiguity and uncertainty which much be resolved by the control process. The DVMT deals with this uncertainty through the use of an opportunistic, goal-directed blackboard architecture. A goal-directed blackboard system involves an extension of the typical HEARSAY II blackboard architecture to include a goal blackboard. Goals are used to focus problem solving through subgoaling while maintaining the advantages of opportunistic data-directed control common to blackboard systems.

Goals are created on the goal blackboard by the blackboard monitor in response to the changes on the data blackboard–e.g., the creation of hypotheses. Goals explicitly represent the system's intention to create hypotheses with particular attributes. The insertion of goals on the goal blackboard results in the planner instantiating Knowledge Sources (KSs) which might achieve the goals. The planner executes a KS's precondition procedure to estimate whether the KS is likely to generate hypotheses to satisfy the desired goal. Hypotheses are created by executing the Knowledge Source Instantiations (KSIs). Knowledge Sources (KSs) are provided to abstract hypotheses from one blackboard level to the next, create tracks from location hypotheses, extend tracks, and merge overlapping tracks. KSs are also provided for internode communication of hypotheses and goals as part of distributed problem solving.

Goals, KSIs, and hypotheses are assigned numeric ratings as they are created. Goal

ratings are based on the ratings of the hypotheses which stimulated the creation of the goal and on the ratings of supergoals (goals which have the goal being rated as a subgoal). KSI ratings reflect both data-directed and the goal-directed components. A KSI rating is a weighted sum of the rating of the goal which the KSI is to accomplish and the estimated rating of the hypothesis the KSI will create. The scheduler uses this formula to rate KSIs on the agenda and selects the most highly rated KSI for execution at each system cycle. Hypotheses are rated as they are created by the executing KSs. The knowledge for producing these ratings is one of the major engineering aspects of the testbed. Though KSs may be independent in principle, the ratings functions associated with the KSs must be consistent with each other if effective control is to result. Thus the reasoning about control decisions is really being done during the engineering of the system rather than during the running of the system.

As we've discussed earlier, systems which use numeric ratings are unable to explicitly consider the evidence implicit in their numeric ratings. They cannot reason about which actions are best for resolving their uncertainty since essentially all they know is the *amount* of their uncertainty–not the *source* of the uncertainty. Much of the research on control for the DVMT has dealt with focusing to avoid distraction from noisy and incorrect data (e.g., data due to ghost tracks and sensor failures). Since the likelihood of potential sources of uncertainty in particular situations cannot be explicitly considered, these focusing mechanisms involve relatively unsophisticated, uniform methods. The DVMT is also unable to reason about the relationships between actions and so may waste processing resources successively pursuing actions which have the same purpose. For example, there are typically a number of sequences of actions which can be used to extend a track hypothesis. Failure of one approach (e.g., due to missing or garbled data) suggests that the other approaches will also fail if they are seeking the same sorts of evidence. What is needed is an action which will seek different sources of evidence to resolve the uncertainty. Processing may also be wasted accumulating less critical evidence. A group-level hypothesis may be pursued by interpreting additional signal data prior to examining more crucial evidence of the group's possible inclusion in a vehicle track. Control should be able consider the purpose of actions in relation to the goal–i.e., producing evidence of complete tracks.

A good deal of effort has been expended developing systems for understanding and explaining DVMT activities because numeric representations of evidence hide much of the problem solving activity. It is difficult to determine why data was or was not used and why hypotheses are or are not believed (beyond meaningless restatements of the ratings). This sort of knowledge is important for users to have confidence in the system's answers and to detect problem solving errors or situations which call for additional problem solving knowledge. Since the ratings play such an important role in control it is not surprising that they do not even really represent belief in the hypotheses, but include

focusing information as well. For example, (potential) track extensions are always rated more highly than their base tracks when they may, in fact, be less certain than a well-supported base track. Finally, the simplistic numeric scheme being used is incapable of representing more sophisticated evidential concepts such as uncertainty and conflicts. This leads to the goal satisfaction problem: how to determine when the system is done–that is, when it has found all the answer hypotheses.

A distributed approach to vehicle monitoring increases the need for intelligent control which reasons about evidence and the best ways to obtain it. In a distributed problem solving environment, no node has access to all of the signal data necessary to be able to interpret vehicle movements. This requires communication between the nodes to request and receive necessary data and evidence. However, communication involves cost, so it is important for such systems to request and transmit only the most appropriate data for reducing the overall interpretation uncertainty.

4.3 Other Plan Recognition Systems

A review of the AI literature reveals that other plan recognition work suffers from the same limitations that our research addresses. In the all of the examples systems discussed in this section, plan recognition is intended to provide an understanding of human goals and intentions based on descriptions of actions such as would be available from a natural language comprehension system.

Schmidt, Sridharan, and Goodson [45,46] present their plan recognition process as a model of how humans understand others actions. Their process involves what they call a *hypothesize and revise strategy.* This approach is motivated by their belief that humans "do not use a strategy of heuristic search to explore a large space of possible interpretations of a sequence of actions," but "explore only a few, usually only one, hypotheses at a time" and are able to adapt the hypothesis to the observations "by a process of refinement and revision." While this sounds very similar to our emphasis on revision, their use of revision is very different. In their system, plans are very general structures which can account for a large number of activities. As actions are interpreted, the revision process uses rules indexed to classes of constraint violations to customize the instantiations of general plans to the particular context. For example, a plan template for making and eating food will not contain any specification of the particular implements to be used nor the possible sequences of actions to obtain and prepare the food. Action observations are used to bind object variables to the particulars of the situation and to insert action hypotheses as appropriate to the goals, subgoals, and prerequisites of the plan instantiations. This makes it possible to deal with alternatives and errors in a way that has not been possible in POISE. However, this generality comes at a cost. More general plans provide far fewer expec-

tations about future actions. Fewer expectations mean fewer constraints and thus greater uncertainty. In the extreme, we could imagine this approach being taken with a single completely general DO-ACTION plan consisting of an indeterminate number of substep DO-ACTIONs. The result would be what [46] terms "postdictive" plan recognition. Such an approach is inappropriate for plan recognition applications which require expectation information–such as an intelligent assistant. If more specific plan templates are to be used, however, there are several problems which must be solved, but which are not addressed in [45,46]. In particular, this system relies on the initial plan instantiation selected being able to be modified to account for later actions. In general, though, there will be multiple relevant alternative instantiations which must be considered. This work has no method for selecting the correct initial plan instantiation (focusing), no method for evaluating belief in alternative instantiations, and no method for shifting between plan instantiations should later action observations completely invalidate an alternative.

In work by Wong [53], plan recognition is used to establish the context within which actions described by English sentences are taking place. Contexts are hierarchies of plan/script instantiation frames which can be used to fill in information not made explicit in the English text. This work does address the problem of selecting the appropriate contexts to some degree. Unfortunately, no explicit representation of evidence and uncertainty is used so the "best validated" context instantiation is chosen based on an ad-hoc heuristic measure. Contexts are deemed more likely when they involve the fewest number of new plan instances and the fewest number of statements to establish context (links from action descriptions to context instantiations). This heuristic is somewhat similar to the "continued vs. started" heuristic in the existing POISE system, however this system has no facility for reconsidering and revising its context interpretations. It can only perform what Wong terms "first-impression" recognition as opposed to "contemplative" recognition. Wong recognizes that this is a serious problem when initial context clues are weak or when the initial context suggested is wrong and must be revised as additional data is accumulated. Finally, there is little control in this system over the instantiation of scripts. The recognition process is bottom-up from an action to all possible contexts–existing and newly created. This approach is infeasible when a large number of potential contexts are suggested as was discussed in connection with POISE.

Work by Allen [2] once again deals with the interpretation of natural language in terms of the goals and intentions of human agents. Allen's system views utterances as speech acts: actions as part of a plan. Rules about likely inferences are then used to drive inferences from the utterance toward the goals and intentions of the speaker. The search is considered to be through a space of partial plans with potential actions in each state being represented by the applicable plan inference rules. Control is accomplished by rating the alternative partial plans based on a number of heuristics. These heuristics refer

to domain-independent relations between plans, their subplans, preconditions, and effects. Ratings are produced from the heuristics using an ad-hoc system of weights. The system is not a general plan recognition system as it is designed to work from a single utterance rather than a sequence of utterances (although work to extend the framework has been done since [38]). Because of this, there is never any need to reconsider interpretations. This is fortunate since the system cannot support revision due to the ad-hoc and implicit nature of the control rating scheme. Finally, the system relies on there being a very small number of plans or goals which the user might be attempting to accomplish–the problems and uncertainties of indexing into a large set of plans is ignored.

Work by Kautz [33,34] is concerned with the development of a general plan recognition system. Kautz uses a logic framework to specify plans in terms of a decomposition hierarchy, a specialization hierarchy, and temporal and parameter constraints. Closed-world assumptions are applied to the action hierarchy to produce the action taxonomy: a complete description of the ways actions can be performed. Plan recognition is then viewed as deductive inference based on the axioms representing the observed actions and the action taxonomy. The framework handles incomplete and missing data in the sense that permissible interpretations can still be determined. It cannot, however, reason about the data in order to resolve uncertainty over partial or conflicting interpretations and so cannot deal with data which is actually incorrect. This work is complementary to our own for it provides a precise, formal semantics for plan recognition in terms of permissible deductions. It does not provide the basis for a practical plan recognition system, however, since it lacks a framework for including the control knowledge necessary for making only likely deductions and interpretations. The only focusing knowledge which the system can apply is the so-called "simplicity constraint." This heuristic closely corresponds to the POISE "continued vs. started" heuristic in its minimization of potential hypotheses although here it is given a formal semantics in terms of circumsription. Kautz makes much of his system's basis in logic and freedom from "probabilistic inference." However the simplicity constraint represents heuristic knowledge which is applied without any explicit representation of its application or its effect on the system's conclusions. Furthermore, since no general purpose theorem proving techniques are capable of handling the inferences in this system, the plan recognition system which is proposed for implementation has a very different flavor from the formal work. In fact, the approach which is proposed is very similar to early POISE control schemes: all of the potential top-level plans which might be supported by the observed actions are created, ranked, and pruned to cover the actions. All observed actions are automatically driven up to all top-level actions through all disjuncts without any application of control knowledge. Focusing decisions are made implicitly through the ranking and covering operations and must consider all possible interpretations of the data at every cycle.

4.4 Plan Recognition Requirements

In this chapter a number of plan recognition systems have been examined and their deficiencies discussed. This final section recaps the major problems that must be solved in order to develop intelligent plan recognition systems. The next chapter will introduce our approach for solving these problems. We feel that an intelligent plan recognition system must be able to:

- Evaluate the level of belief and uncertainty in alternative interpretations.
- Understand the reasons for beliefs.
- Encode and apply heuristic control knowledge.
- Revise interpretation hypotheses as information accumulates.
- Handle uncertain and incorrect data.
- Integrate data from multiple sources.
- Actively control data accumulation.
- Reflect system goals in control decisions.

Of the systems examined, only the DVMT has any ability to evaluate its level of belief in interpretation hypotheses. A single number representation of belief cannot differentiate between belief and uncertainty, however, and does not provide access to any of the reasons for the beliefs. The POISE focusing system does contain a representation of its reasons for making focusing decisions, but does not provide any measure of the belief or uncertainty of the alternative hypotheses. Use of an independent evidential reasoning system has many advantages. Knowledge of belief and uncertainty can be used by the focusing and revision processes to develop efficient and sophisticated control schemes. Control need no longer be limited to pursuing only the most highly rated/believed hypotheses, but may reason about the best actions to take given the existing interpretations and data. Reasons for beliefs can be used to justify system interpretations to users and to facilitate analyses of system performance. Existing plan recognition systems cannot explain why they believe the current hypotheses, why they are still uncertain about them, and why they chose to perform particular actions.

One of our major concerns in earlier work was the development of a framework for representing and applying heuristic knowledge for focus-of-attention. This is an important issue because of the ambiguity inherent in many plan recognition applications. It is computationally infeasible to use a brute force approach and pursue all of the potential

interpretations. Decisions must be made about which interpretations and data to devote the system's limited processing resources to. The sort of commonsense and expert knowledge that can provide this focusing is available if there is an appropriate framework within which to encode and apply it. The existing systems that make use of heuristic focusing knowledge have inadequate mechanisms for dealing with the amount of knowledge we envision using.

It should be noted here that we intend heuristic control to be extended to all aspects of the interpretation process. In existing systems, little attention has been paid to controlling the actual steps in building potential interpretations. In POISE and the Kautz' system for example, data is abstracted to top-level plan instantiations before any sort of focusing intelligence is applied. While this may be acceptable for some domains, it is not in general. In the software engineering domain for POISE, it is easy to recognize lower-level plan instances such as editing a file, but difficult to determine what top-level plan instance these actions may be part of. This is because of the weakness of the constraints and the large number of disjunctions in the plan library (editing a file can be a part of nearly every plan). In such situations, constructing all possible interpretations for each action and then eliminating those deemed less likely is impractical. Instead, the system needs to reason about whether it is appropriate to abstract the current interpretations further based on the degree of uncertainty over what they represent.

Any focusing scheme excludes interpretation alternatives based upon uncertain, heuristic knowledge. Since additional evidence may show that incorrect focusing decisions were made, such a scheme must include provision for revising its decisions and reconsidering abandoned alternatives. Both POISE and the Kautz' system provide some revision capability. In the case of Kautz, since focusing consists of applying a single heuristic there is little reasoning that can be done during the revision process. In POISE the problem is that it is difficult to integrate the new knowledge with existing focusing knowledge to control the revision process. Thus revision suffers from a lack of control knowledge for focusing the revision process.

None of the example systems handle uncertain data in a satisfactory way. Data may be uncertain because of noise or errors or because data is missing. Handling uncertain data means more than simply making those inferences which are possible given missing data or ignoring data which seems to be noise since it cannot be satisfactorily interpreted. An intelligent plan recognition system should be able to reason about the nature of the uncertain data–e.g., what data is missing or how that data has been garbled. In order to fully comprehend a system's interpretations, users must have access to knowledge about assumptions that underlie the interpretations.

One approach to handling uncertain data is to be able to integrate multiple forms of evidence to reach interpretations. Existing plan recognition systems have limited their

evidence to that obtained from a single main source of input data. True vehicle monitoring situations will require the integration of data from different types of sensors and other intelligence sources. Just as there is knowledge beyond that contained in the plans which can be used for control and focusing, there typically is additional knowledge which can used as evidence for or against interpretation hypotheses. This sort of domain knowledge is available to expert users to help resolve interpretation uncertainties. Such domain knowledge, termed "first-principles" knowledge, has been investigated for its role in human understanding of software engineering tasks.

In addition to the problems discussed above, there are two more areas of concern which are best considered extensions to the view of plan recognition used by the existing research systems. In any actual plan recognition application, the overall system will have some purpose. Rather than use completely different systems tailored to the application it should be possible to make the operation of the system sensitive to different goals. For example, an aircraft monitoring system might be used for air traffic control and it might also be used for military monitoring. The purpose of military monitoring applications may be for the protection of certain installations. In this case it would be less important to form complete models of the environment than it would be to focus on aircraft which could pose a threat. Thus the goal of protecting the installation should be able to influence the interpretation process of the aircraft monitoring system.

Another extension involves active control over the gathering of data. In vehicle monitoring, for example, sensors might be under the control of the interpretation system. The system could then adjust sensor characteristics to gather the kind of data that would best resolve its uncertainties. Active control might also be used in an intelligent assistant such as POISE by allowing the system to interact directly with the user to gather information. Active control of data gathering could greatly enhance the efficiency of an interpretation by allowing the system to gather the most useful data. The degree to which this is possible will depend upon the domain. In an intelligent assistant there would be only limited opportunities for the system to actively pursue information.

Chapter 5

Evidence-Based Plan Recognition

In this chapter we introduce our view of evidence-based plan recognition. The first section outlines the basic elements of the approach and explains how they allow us to address the plan recognition requirements of section 4.4. In the Taxonomy section, the various kinds of knowledge necessary for such a system are examined along with instances from the POISE and DVMT domains. Finally, an example of evidence-based plan recognition using the vehicle monitoring domain is presented.

5.1 The Approach

By evidence-based plan recognition, we mean that plan recognition should be treated as a process of *gathering evidence* to *manage uncertainty*. Uncertain interpretation hypotheses must be "proved" by collecting appropriate additional evidence. This is a unique view of the plan recognition process. Typically, plan recognition systems have used a "constraint satisfaction" approach, with the input data providing the interpretation constraints. Of the systems that were reviewed in chapter 4, only the DVMT could be described as accumulating evidence. The DVMT uses an extremely limited concept of evidence, however, and does not make explicit decisions about the uncertainties its actions are intended to resolve. Extending the representation of evidence and using this representation as the basis for control decisions makes it possible to develop a framework within which the *purpose* of actions can be understood. Interpretation actions are taken in order to resolve particular sources of uncertainty. Thus, the most appropriate action to take depends upon the most "desirable" uncertainty to try to resolve and the "best" action to resolve it. A system which reasons about its control decisions in this way can truly *gather* evidence to *manage* uncertainty rather than just *accumulate* evidence to *resolve* uncertainty. The distinction results from the existence of multiple system goals and the need to consider the tradeoffs during the control process. The immediate effect of an action must be weighed against the long term role of the action as well as other concerns such as execution time and safety.

T··king this view of plan recognition, it is possible to develop a system which meets the requirements outlined in section 4.4. The key characteristics of the approach are:

- Plan, subplan relations are treated as uncertain, evidential relations.
- Evidence and sources of uncertainty are explicitly represented.
- Heuristic control decisions are based on the sources of uncertainty in the hypotheses and the need to manage uncertainty.

Treating plan, subplan relations as evidential relations simply means that we treat these relations as resulting from uncertain inferences. By way of comparison, the constraint satisfaction approach to plan recognition treats these relations as absolute grammatical relations. Thus, instead of saying that a (grammatically correct) subplan instantiation *satisfies* a subgoal of a plan instantiation, we say that it is *evidence* for that plan instantiation. This gives us a better framework for dealing with the uncertainties inherent in plan recognition. It now makes sense to use an evidential reasoning system which can evaluate the level of belief and uncertainty in hypotheses based on the level of belief and uncertainty in the evidence. Uncertain or incorrect data can easily be accommodated since this possibility simply results in additional uncertainty for hypotheses relying on such data. Multiple sources of evidence can be integrated because all evidence is treated in a uniform fashion–i.e., as evidential inferences. Revision of interpretation hypotheses is accommodated within an evidential reasoning framework because such a system is inherently nonmonotonic–the belief in the alternatives changes as evidence is accumulated. "Inconsistencies" are represented as contradictory evidence and as such are simply another source of uncertainty to be resolved.

By an explicit representation of evidence, we mean that evidential links are explicitly maintained between symbolic representations of evidence and the representations of the hypotheses that the evidence supports. This approach makes it possible to reason about more aspects of the evidence than simply its "strength." Knowledge of the kinds of evidence underlying beliefs makes it possible to understand the *sources of uncertainty* in the beliefs. Evidence provides uncertain support for a hypothesis because there are conditions under which the evidence may fail to support the hypothesis. Numeric rating functions gathered from experts typically summarize just such knowledge–along with a priori likelihood judgements. Explicit information about the sources of uncertainty in evidence is a type of knowledge that we feel is very important for plan recognition. One of its advantages is that it makes it possible to evaluate belief in specific contexts rather than having to rely on general, a priori judgements. For example, missing data in a track of limited length is an important source of uncertainty while missing data in a very long track is insignificant.

The major advantage of sources of uncertainty knowledge is that it provides the perfect basis for making control decisions. Since the goal of plan recognition is to "prove" hypotheses by resolving uncertainty, a sophisticated control process must be able to understand the sources of uncertainty in the hypotheses and reason about the best actions to take to resolve them. Thus sources of uncertainty are used to elucidate control choices by a process which determines which goals remain unsatisfied due to excessive uncertainty, what the sources of that uncertainty are, and what actions could provide evidence to resolve the uncertainty. Note that the "actions" we refer to here involve the way the interpretation system pursues its hypotheses as opposed to the "domain actions" that the system may be trying to interpret. Interpretation actions include the interpretation of data and hypotheses to produce evidence for higher-level hypotheses. Managing uncertainty means that the system must be sensitive to the various goals of the application. Since the purpose of the different actions is now understood, a variety of factors can be weighed during the decision process. For example, the facility protection goal in a vehicle monitoring application results in increased importance being placed in resolving any uncertainty in the *identity* of potentially hostile vehicles. Active control of data accumulation is also easily accommodated in a system which reasons explicitly about uncertainty. Understanding what information would be most effective at reducing its uncertainty, such a system might choose to actively gather the evidence rather than waiting passively for data to constrain its interpretations. In domains where this is possible, the set of interpretation actions could then be extended to include methods for interacting with the environment in order to affect the data which is gathered.

Sources of uncertainty information also provides an ideal framework for specifying and applying heuristic control knowledge. Heuristic focusing and control knowledge simply represents expert knowledge about methods for dealing with uncertainty. POISE focusing heuristics, in effect, specify what alternatives to gather additional evidence for based on implicit decisions about evidence and uncertainty represented in the rule antecedents. For example the "continued vs. started" heuristic is really saying that it is best to look for additional evidence for the in-progress hypothesis since, implicitly, it has accumulated more evidence. Using a system for representing evidence and uncertainty, such a heuristic can be generalized in two stages. First, we can imagine a heuristic which says to "pursue the most believed hypothesis." This form is advantageous because there no longer needs to be a large collection of heuristic focusing rules which might offer conflicting advice. The same level of reasoning must be done, i.e., figuring out which alternative has the most evidence for it, but this reasoning can be done in a more logical way as part of the evidential reasoning system. The problem with this form of the heuristic is that it still requires qualifications specifying exactly when it's *not* best to pursue the most believed hypothesis. This is because pursuing the most highly believed alternative–while usually the best way to gather evidence–may

not always be the most effective way to resolve particular interpretation uncertainties. Selecting the correct decision depends upon an understanding of the purpose of the action as well as the characteristics of the situation. Thus an alternative method for capturing this focusing knowledge is a scheme for deciding what actions can potentially provide the best evidence based on resolving the relevant *uncertainties.* Again, it's not that any less knowledge is needed to be able to reason effectively in this way, it's simply that a framework keyed to interpretation uncertainties provides a logical, modular framework for the specification and application of expert focusing knowledge. Actions are taken for the purpose of resolving particular uncertainties so it makes sense to identify which actions are best for which uncertainties.

5.2 Taxonomy

In this section we present a taxonomy of the knowledge that must be part of an evidence-based plan recognition system. This taxonomy represents a first attempt at generalizing and categorizing the knowledge being investigated for the research domains. In addition to general descriptions of the knowledge, example knowledge from the POISE and DVMT domains is included.

5.2.1 Hypotheses

Hypotheses represent plan instantiations for which evidence has been gathered–i.e., for which we have some level of belief or disbelief. In order to control the creation of hypotheses we must be able to represent just the level of plan hypothesis that the evidence actually supports. This means that hypotheses at any level of abstraction must be treated in a uniform fashion. Most existing plan recognition systems force hypotheses to be abstracted to top-level plans–through multiple levels of uncertain disjunctions–regardless of whether there is sufficient evidence to guide the process. This is because of the special role top-level plans play in these systems. Hypothesis uniformity is also important for the integration of evidence from a variety of sources since this evidence might support hypotheses at any level of plan abstraction.

Hypotheses are complex structures. They include evidence and parameter information. Evidence is represented by an explicit link between the evidence and the hypothesis it supports or detracts from. Because of the uniformity requirements, hypotheses consistent with the constraints of a higher-level hypothesis are represented as just another source of evidence for the higher-level hypothesis–rather than as part of some special grammatical relation. However, the evidential links must still include information about the type of evidential inference represented and the role that the various pieces of evidence play in the inference (see section 5.2.2). The sources of uncertainty in each evidential inference

are represented as symbolic tags on the evidential links. Sources of uncertainty are not propagated, but are accessible by tracing the evidence hierarchy. Uncertainty in the exact values of hypothesis parameters is represented by maintaining the range or set of possible values.

One of the key characteristics that distinguishes plan recognition from diagnosis problems is the fact that evidence not only *justifies* the plan recognition hypotheses, it also *refines* them (see chapter 3). That is, we don't simply gather evidence to decide our belief, we gather it to define exactly what it is we believe as well. Plan definitions not only specify what is valid evidence for the plans, but also specify how the plan parameter values are related to the characteristics of the evidence. For example, in vehicle monitoring, group-level hypotheses not only provide evidence for vehicles, but also define the vehicle type and position. One consequence of the fact that evidence refines hypotheses as it supports them is that the addition of evidence generally modifies a hypothesis. This leads to the need for multiple "copies" of a developing hypothesis and a control process which considers when to deal with uncertainty by making copies. Uncertainty is no longer simply a question of whether or not evidence supports a hypothesis, it is also now a question of exactly what the correct hypothesis is.

5.2.2 Evidence

By evidence, we mean the reasons to believe or disbelieve hypotheses. As discussed earlier, we view subplan instances as (uncertain) evidence for plan instances. Rather than summarize the "quality" of this evidence numerically, however, we maintain explicit links between the evidence and the supported hypotheses. This makes it possible to understand why the evidence fails to conclusively prove the hypotheses by giving us access to the sources of uncertainty in the evidence–that is, the reasons the evidence may fail (see section 5.2.4.).

Viewing plan, subplan relations as evidential relations means that we view plan definitions as specifications of the valid evidential inferences, $\{A_i\} \Rightarrow B$, where the A_i are sources of evidence and B is a plan. Plan inferences are uncertain, evidential inferences rather than deductive inferences because the existence of the evidence, $\{A_i\}$, is not sufficient to guarantee the conclusion, B. These inferences are in fact a form of *abductive* inference [7]: if x causes y and y is true then hypothesize x. Plan definitions can be viewed as statements that if plan B occurs then the data $\{A_i\}$ will (be caused to) occur. Thus B is an explanation for $\{A_i\}$. Of course, abductive inferences are merely plausible inferences as there may be other explanations for the same data. In addition to this basic uncertainty, plan recognition inferences are uncertain for other reasons as well. For instance, plans must typically be inferred based on incomplete, partial evidence. That is, plan B may be hypothesized based on evidence $\{A_j\}$ where $\{A_j\} \subset \{A_i\}$. A complete discussion of uncertainty may be found in section 5.2.4.

There are two ways to go about resolving the uncertainty in plan hypotheses resulting from these evidential inferences:

1. Gather additional evidence to directly resolve the uncertainty in the inferences.

2. Gather independent evidence for the supported hypotheses.

Explicit representation of the evidential links makes it possible to understand what the remaining ***sources of uncertainty*** are in an inference and gather evidence to resolve these sources of uncertainty. For example, a hypothesis B, based on the evidence $\{A_j\}$ (where $\{A_j\} \subset \{A_i\}$ above) is uncertain because only part of the conditional evidence is being used to make the inference. Thus, uncertainty in B could be resolved by accumulating the rest of the evidence $\{A_i\}$. There are typically a number of sources of uncertainty for each source of evidence. Each of these sources of uncertainty will then have a characteristic set of sources of evidence which can be used to resolve the uncertainty.

Independent evidence for this same hypothesis B, requires that there are additional ways to infer the plan, e.g., $\{D_i\} \Rightarrow B$. Then the evidence $\{D_i\}$ can be used to lend further support to B and so resolve uncertainty in the hypothesis. It should be noted that evidence is not necessarily independent just because it is based on a separate inference axiom. Instead, what must be true is that the independent evidence must not include the same sources of uncertainty as the old evidence. For example, if evidence from radar scanning and radio emissions detection can both be affected by the same kind of weather conditions (and affect the interpretation in the same way) then one could not be used to resolve uncertainty based on the other.

In POISE and the DVMT, plan inferences are based on partial conditional evidence. A plan for Purchasing Items may be inferred from the occurrence of a Receive-Purchase-Request plan in POISE. Likewise, a vehicle may be inferred from a single group hypothesis. In each case, the recognition systems pursue these hypotheses by trying to complete the set of subplan instantiations for the hypotheses. Neither POISE nor the DVMT use multiple sources of evidence which could be used to develop independent evidence for the hypotheses. For example, POISE could interrogate the user as to the correctness of the plan for Purchasing Items while the DVMT could make use of data from other types of sensors to support vehicle hypotheses.

As well as the supportive evidential inferences described above, the system must also handle negative evidential inferences–i.e., inferences which detract from belief in hypotheses. Some negative evidence may result from what we term direct negative inferences–that is, from inference axioms of the form $\{A_i\} \Rightarrow \neg B$. These inferences are not based on axioms which result from the standard plan definitions, but on additional evidential axioms which may be specified. Most negative evidence, however, results from inferences which are based on the standard plan definitions. Viewing the plan definitions as statements that

if plan B occurs then the data $\{A_i\}$ will (be caused to) occur, it is clear that this is equivalent to inferences of the form: $\neg\{A_i\} \Rightarrow \neg B$. Thus evidence gathered for an alternative interpretation of evidence A_k in an inference $\{A_j\} \Rightarrow B$ acts as negative evidence for the plan hypothesis B.

One issue for systems which use explicit, symbolic representations of evidence is how to go about evaluating the level of support provided by the evidence. The level of support provided by evidential inferences depends on the likelihood of the sources of uncertainty holding true. In conventional numeric evidential reasoning systems, this evaluation is made using a priori knowledge and is fixed in a degree of belief rating. While the exact same information that is used in the numeric combining functions could be used to evaluate the explicit evidential representation, it is possible to do much better if additional knowledge can used to examine the particular situation. The likelihood of inferences can in part be evaluated based on the strength of the constraints which validate the inference. The strength of constraints varies with the particular inference axiom and with the premise evidence, i.e., $\{A_j\} \subset \{A_i\}$ as above. For example, each successive vehicle track extension provides additional support for a vehicle track hypothesis. That support is not uniformly additive, however, because little or no support is provided until the number of correlated signals reaches some sort of minimum, support provided by a single extension is much smaller than the corresponding fraction of belief in a fairly long track, and the length of complete tracks varies. Belief is not just some more complex, fixed function of track length either as it would be in a numeric reasoning system. This discussion highlights the differences between the sources of uncertainty and the factors which can used to evaluate likelihood. Sources of uncertainty are the reasons that evidence may fail while uncertainty factors can be used to judge the likelihood of failure.

5.2.3 Data

We use the term data to refer to externally generated information which can serve as a source of evidence for interpretations. The other sources of evidence are the plan hypotheses themselves since they provide evidence for higher-level hypotheses. All sources of evidence must be processed, or "interpreted," before they become evidence. This involves evaluating the plan constraints and deciding how to represent any uncertainties. For example, radar data may support an existing vehicle hypothesis or it may support a new vehicle hypothesis. The exact representation of this uncertainty is a heuristic control decision which must be made during the interpretation process.

Both POISE and the DVMT use only single sources of interpretation data. In POISE, data consists of primitive plan hypotheses representing abstracted views of the actions taken by the user. The recognition of tasks then requires the interpretation of these primitives through several levels of plan hypotheses. Data in the DVMT consists of signal

hypotheses which simulate the result of low-level processing of acoustic sensor output. Each signal hypothesis represents a perceived environmental source and includes the following information: frequency, position, time of detection, and a numeric belief rating. Data must be interpreted through several levels in the plan grammar before it provides evidence for vehicle track hypotheses.

While the DVMT has used small amounts of simulated data, a real-world vehicle monitoring system would generate such large amounts of data that it would be infeasible to consider all of the data. However, since most of this data results from noise or irrelevant signal sources, failing to examine and interpret all of the data can have little effect on the uncertainty of the system. This is in marked contrast to POISE where all data must be interpreted even if it turns out to be an erroneous action on the user's part. This suggests the need for some kind of system parametrization to account for the role that data plays in driving the interpretation process. In the DVMT, the examination of data must be strictly controlled based on its potential for providing evidence for the developing interpretations. User actions in an intelligent assistant, on the other hand, drive the interpretation process. The different roles that data plays in different applications can be accounted for through the use of system goals (see section 5.2.7).

5.2.4 Uncertainty

Most evidence is inconclusive. That is, it does not confirm or disconfirm a hypothesis, but merely supports or detracts from the hypothesis. The reason for this is that the evidence is uncertain–there are conditions under which the evidence may fail to support the conclusion. These conditions–the reasons that evidence can fail–are what we term the *sources of uncertainty* in the evidence. One of the important advantages of our approach is that we make it possible to understand exactly what the sources of uncertainty are in the evidence for a hypothesis. This allows the control process to reason about the best ways to resolve the system's uncertainty by considering the reasons for that uncertainty.

We view plan definitions as specifications of the valid evidential inference axioms for the plan recognition application. The form of these axioms is, in effect, $\{A_i\} \Rightarrow B$, where the A_i are sources of evidence and B is the supported plan. Inferences based on the axioms are uncertain, evidential inferences rather than deductive inferences because the existence of the conditional evidence, $\{A_i\}$, is not sufficient to guarantee the conclusion, B. Hypotheses are further compromised by the fact that inferences must often be based on partial evidence. That is, a plan of type B will typically be hypothesized based on evidence $\{A_j\}$ where $\{A_j\} \subset \{A_i\}$. Partial evidence may be used because the application requires incremental plan recognition for real-time recognition or because of computational considerations such as the complexity of evaluating the constraints to recognize valid inferences.

Given the hypothesis B based on the inference $\{A_j\} \Rightarrow B$, where the axiom is $\{A_i\} \Rightarrow B$

and $\{A_j\} \subset \{A_i\}$, there are the following potential classes of uncertainty sources in the hypothesis:

- The premise evidence may be uncertain–i.e., some A_k is uncertain because it is also based on uncertain evidence.
- There may be alternative interpretations for the evidence–i.e., for some $A_k in\{A_j\}$ the correct inference is $A_k \Rightarrow C$.
- The inference may be based on partial premise evidence–i.e., $\{A_j\} \neq \{A_i\}$.
- It may be uncertain whether the evidence satisfies the inference axiom (meets the constraints)—that is, it is uncertain whether $\{A_j\} \subset \{A_i\}$.
- The inference axioms may themselves be uncertain–that is, the correctness of $\{A_i\} \Rightarrow B$ is uncertain.

The actual instances of sources of uncertainty for each source of evidence in a domain fall into these classes. For instance, acoustic sensor data in the DVMT provides uncertain support for an environmental signal hypothesis because it may result from sensor malfunction. That is, a source of uncertainty in the evidential link between acoustic sensor data and a signal hypothesis is the potential alternative interpretation of the data as resulting from a sensor malfunction. Likewise, a signal hypothesis may support a group hypothesis (and so eventually a vehicle hypothesis), but it may also fail to support it because the signal actually is noise–the correct (alternative) interpretation of the signal is as noise rather than as part of a group. A group hypothesis may support a vehicle hypothesis, but until the complete set of groups for the vehicle support the vehicle hypothesis, it is uncertain. Because of sensor resolution, the exact values of acoustic data parameters such as position and frequency are uncertain. This can result in uncertainty over whether a particular evidential inference is valid. Track hypotheses may represent actual vehicle tracks, but might also result from correlated noise or ghost data. While with likelihood of the track increases as the length is increased, there is always some degree of uncertainty from the fact that there is no set definition of a "complete" track. POISE never considered data errors, but a real-world intelligent assistant would have to be able to deal with user errors. User errors can be handled by making them a source of uncertainty in the data.

In a sophisticated plan recognition system with access to many sources of evidence, sources of uncertainty are used to drive the control process by selecting actions which result in evidence to resolve the important sources of uncertainty in the interpretations. For example, uncertainty due the possibility of sensor malfunction may be resolved (or at least partially resolved) by ordering diagnostics to be run on the sensor. Noise due to weather or terrain factors might be ruled out by checking weather conditions and the terrain. Of

course, uncertainty due to partial premise evidence is one of the most prevalent sources of interpretation uncertainty and one which is not terribly amenable to sophisticated evidence gathering. However, because this source of uncertainty can be explicitly considered it may still be possible to improve evidence gathering: some incompleteness may not be very significant in the overall support of a hypothesis, the support provided by various (still incomplete) extensions may be stronger than others, and alternative, independent evidence may provide a less expensive resolution.

The sources of uncertainty in an evidential link may be represented in different ways depending on the type of uncertainty and what is most appropriate for future action. One way is to do it in a manner similar to the parallel-certainty inference approach (see chapter 3). Attached to the evidential links are symbolic statements which qualify the links with information about why they are uncertain. When uncertainty results from alternative interpretations, it may be more desirable to represent the alternative interpretations explicitly as hypotheses. Alternative interpretations are then connected with the links which specify that they are alternatives. This makes it easier to reason about pursuing evidence for the alternatives.

As was mentioned in section 5.2.1, hypothesis parameter values can also be uncertain. These uncertainties result from parameter uncertainties in the evidence, e.g., resolution uncertainty of acoustic sensors for frequency and position, and from incomplete evidence with which to define the parameters. Parameter uncertainty is represented by representing the potential range or set of supported values for the parameters. Copying of hypotheses when parameters are refined eliminates the need to understand the exact relationship between the evidence and parameter values since the values need never be revised. However, some understanding of the connections are required to evaluate the relative likelihood of the various values and to choose appropriate evidence to resolve the parameter uncertainties.

5.2.5 Actions

The control process eventually involves a decision about the next action for the interpretation system to take in order to generate additional evidence. Actions may involve the interpretation of existing data or hypotheses. For example, looking for additional evidence for a hypothesis in the acoustic sensor data being automatically collected or abstracting previously created hypotheses to check if they support a valid answer. These interpretation actions must be distinguished from the domain "actions" that the system is trying to interpret. Interpretation actions do not involve an interaction with the environment. However, in some domains, evidence-gathering actions may be extended to involve active processes to accumulate external data. For example, a vehicle monitoring system might be able to make specific requests for data from particular sensors. It depends on the nature of the domain as to how active the interpretation system may be in gathering evidence.

In some applications such as intelligent assistants, the ony "action" that the system may be able to take is to wait for more data to be generated and provided to it.

The actions being referred to above are what we term evidence-gathering actions since they involve the generation of evidence for the interpretations. The system also takes other actions–control actions–as part of the process of making a decision about which domain action to take. Control actions include the identification of additional evidence relevant to resolving a partial evidence uncertainty and locating relevant data to be used to develop the desired evidence. For further information on control actions see section 5.2.8.

5.2.6 Relations

The plan recognition process involves the creation and consideration of interpretation hypotheses. These hypotheses are not always independent, but are related in that belief/disbelief in one affects the belief/disbelief in another. For example, a hypothesis can be an extension of another hypothesis. Evidence against an extension may or may not be evidence against the original hypothesis depending on whether it is really evidence against the plan instantiation or just against this particular extension of the instantiation. However, evidence against a hypothesis is always evidence against any extension hypotheses. Hypotheses may also be related as alternatives either through the interpretation of the same data or because they are based on inconsistent problem solving situations. In this case, evidence for a hypothesis is evidence against its alternative and vice versa. The existence of alternatives increases the level of uncertainty in a hypothesis.

POISE plan instantiation hypotheses may be related by being alternatives or by one being an extension of the other. Hypotheses are alternatives when they cover overlapping steps and they are not assumed to share the steps. Alternatives may also be recognized based on domain knowledge such as that in the "first-principles" knowledge mentioned above or through dependence on conflicting assumptions. For example, there may be knowledge that makes it unlikely that two plan instantiations could occur simultaneously or that one plan would be carried out soon after another. These relations must be explicitly recorded since they affect belief and uncertainty in the hypotheses and since evidence must be propagated over these relations. When one hypothesis represents an extension of another hypothesis with additional step data they are not alternatives in the sense given above since belief and evidence propagates differently between such hypotheses and since control strategies proceed differently. In particular, extensions enhance belief in a hypothesis, but control will generally proceed to explore the extensions.

Relations between hypotheses play an important role in the interpretation process in a vehicle monitoring system and yet a representation of these relations is absent from the current DVMT. In particular, the existence of a number of alternative extensions for a track hypothesis results in a good deal of uncertainty in the true track position which

can be eliminated by exploration of the alternatives. For example, given a fairly good length track hypothesis with several possible extensions, our belief in the correctness of the track might be high without high belief in any one of the extensions due to the number of alternatives. The best control approach would be to try to reduce the uncertainty in the least expensive manner. This might suggest a breadth-first development of the alternatives rather than the depth-first approach which the DVMT would pursue. The DVMT would proceed depth-first because it has no knowledge of the alternatives and the uncertainty they cause. Instead, any single point extension which succeeds would be rated more highly than the unextended track despite the fact that a single point extension would do little to reduce the uncertainty.

5.2.7 Goals

The primary responsibility of the control component is to try to satisfy the system goals. For example, the overall system goal may be to determine all and only those answers that can be supported by available data. This goal is what we might call an "uncertain goal" since its satisfaction will always be uncertain due to the inconclusive nature of evidence. Instead of saying these goals are *satisfied*, we will say that they are *sufficiently satisfied* or not as evidence is accumulated. What constitutes an acceptable answer is specified by answer and evidence constraints which express the desired system goals. The answer constraints specify the acceptable plan instances including plan types and parameter values while the evidence constraints specify accptable hypothesis and parameter uncertainties. Thus the top-level control process must evaluate the suitability of the existing evidence in terms of the remaining uncertainties in order to determine whether the system constraints are met. If the goals are not yet satisfied, the control component must decide which uncertainties to resolve and how to resolve them.

Since the number of answers supported by the data is uncertain, an integral part of satisfying this system goal is the determination that all acceptable answers have been found. The explicit inclusion of this control goal is important because it affords a natural way of including certain types of evidence and uncertainties which are awkward to apply in the process of constructing answers. It also affords a metric for judging when processing is complete. This is crucial for situation assessment tasks like vehicle monitoring where processing of all data is impossible due to the huge volume of data being generated by many sensors.

Other applications may be handled by changing the overall system goals. An intelligent assistant application such as POISE would require that all data (from the user) be covered by an interpretation. Monitoring aircraft to prevent an attack would mean specifying that answers consist only of attack plans. In addition the system goals may also include resource constraints. For example on interpretation costs such as elapsed time, processing

time, and safety concerns. These goals affect the choice of actions to take by influencing the desirability of goals and actions.

5.2.8 Control Strategies

As was discussed in section 2.1, control decisions are typically made in a two stage process which first identifies actions relevant to satisfying the goal and then chooses the best action to take next. In our approach, relevant actions are identified through a planning process based on the evidence gathering view of plan recognition. This planning process makes use of the explicit knowledge of the *sources of uncertainty* in the hypotheses both to elucidate the relevant control choices. Sources of uncertainty information is important because it is just these uncertainties which must be resolved by the plan recognition process. That is, the *purpose* of the actions taken by the plan recognition system is to resolve these uncertainties. Each type of evidence has its characteristic uncertainties and each type of evidence can be used to resolve particular uncertainties. The planning process begins by evaluating the problem solving goals to identify those goals that remain unmet due to insufficient evidence. These sources of uncertainty in the solution goal are then used to identify the types of domain evidence that should be gathered. Methods for gathering this evidence are then created and refined. Note that control plans should not be confused with the domain plans which the system is attempting to interpret.

Control plans are elaborated to the point of choosing the next "domain action." That is, actions which create evidence about the domain. Plans are generally not elaborated beyond the next step because the outcome of the actions is uncertain. Planning is basically a top-down process which reflects the desirability of classes of actions to solve the problem. However, at the lower levels of the process, planning is driven by the available data and so reflects the feasibility of the actions. This results in a control strategy which integrates both data-directed and goal-directed components. However, control is not "opportunistic" since it does not key on data to select appropriate goals. Typically, opportunistic control has relied on simplistic measures of data "goodness" to select appropriate actions. Since we believe that the "goodness" of data can only be judged relative to a particular goal, opportunistic control would require the capability of recognizing when data was good and what goal it was good for.

Since the planning process typically results in a number of ways in which the system goals can be pursued, an additional *focus-of-attention* process is used to select the single action to take next. This focusing process relies on the encoding of heuristic knowledge about the best ways to pursue goals and gather evidence. We believe that the goal refinement based on sources of uncertainty knowledge provides the perfect structure for applying focusing heuristics. Conventional approaches to heuristic focusing tend to suffer from the problem of conflicting heuristics (see section 2.1). The specification of the heuristics is

so general that multiple heuristics with conflicting suggestions apply at the same time. We believe that this problem can be overcome by using a hierarchical framework–based on the refinement structure–within which to encode and apply the focusing knowledge. Heuristics are then indexed according to the exact class of situations to which they apply. This amounts to extending the heuristics to include sufficient information about the basis of the heuristics to resolve conflicts by understanding which better applies. For example, in a DVMT-like vehicle monitoring system we could imagine the following two focusing heuristics regarding the selection of acoustic sensor data: prefer well-sensed (loud) signal data and prefer data at times with small numbers of clusters (potential sources). While these heuristics may seem to offer conflicting advice in many instances, once we understand the reasoning behind them we see that they actually apply in different situations based on the purpose of the data interpretation (see section 5.3).

Heuristic knowledge is indexed via a tree which is a static abstraction of the potential control plan graphs. Nodes in this tree represent the various sources of uncertainty, evidence, and classes of data which may be encountered in the domain. Heuristics are then associated with the nodes in this tree which represent the situations and data characteristics about which the heuristics make ordering suggestions. The heuristics mentioned above do not conflict because though they both refer to acoustic sensor data characteristics, they do so on independent paths of the heuristics tree. The well-sensed data heuristic, for instance, is associated with the purpose of resolving hypothesis parameter uncertainties like position and frequency–that is, it is associated with acoustic sensor data in the path below the goal of resolving uncertainty in parameters. The cluster heuristic is associated with acoustic sensor data also, but on a different path in the tree associated with resolving hypothesis existence uncertainty. One big advantage of this approach is that heuristic knowledge can be applied *during* the refinement process rather than after it. As plans are being refined, it is possible to examine the heuristics tree at each step and see whether there are any applicable heuristics which would allow us to prune refinement paths. Of course, since it is impossible to tell whether there actually are any actions to carry out selected goals, this sort of pruning requires that the refinement be augmented with a backtracking scheme.

We view focusing as a task for expert-level heuristic reasoning both in selecting the uncertainty to pursue and in determining how to pursue it. Expert knowledge is based on a broad understanding of the domain. It involves knowledge of what evidence can be most easily gathered and how evidence affects the uncertainties in the various goals. Note, though, that control decisions must take into account the long-term effects of the actions rather than just the immediate effects. This is what makes the process so difficult and forces the system to rely on expert-level heuristic control knowledge. It is essentially hopeless to try to make an "objective" cost/benefit analysis because it's impossible to

objectively assess the long-term effects of actions. For example, in a vehicle monitoring system we might reason that it makes little sense to accumulate evidence for a precise track position before accumulating evidence to resolve whether the track represents a vehicle of interest (a potential answer) or not. It may also be that much of the evidence for resolving uncertainty in a vehicle hypothesis will also be useful for resolving vehicle ID and position. From these considerations we could conclude that evidence to resolve uncertainty in whether a hypothesis represents an actual source should be gathered next–even if actions to resolve uncertainty in position might look better in some immediate sense.

5.3 An Example

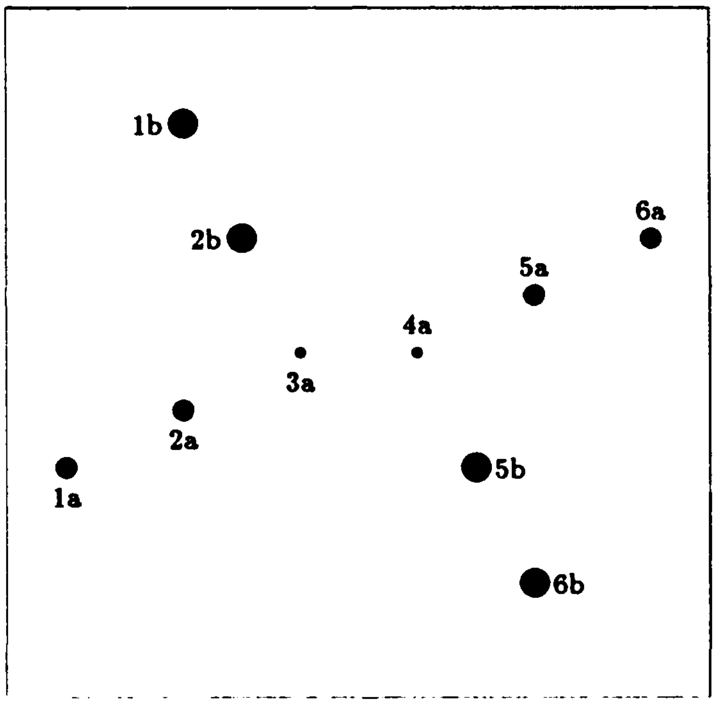


Figure 5.1: Acoustic Sensor Data Clusters

In this section we will present a very simple example which illustrates the basic operations of the sort of system we envision. A problem which is often used with the DVMT [22] will be examined. The acoustic sensor data for this example is shown in figure 5.1. Two potential, intersecting vehicle tracks are contained in the data. Data points 1a through 6a, track a, represent the "correct" track while those of track b represent "ghost" data. The plotted data points consist of clusters of signals. Clustering is common practice for facilitating the handling and analysis of acoustic sensor data [24] and this approach has recently been taken with the DVMT [22]. In fact, there will typically be various types of

automatic, low-level processing which can be done on raw data to assist the interpretation process. For example, clustering tends to group signal data from a single source and hence can guide the selection of data for interpretation when the task is to confirm the existence of a given potential source. Additional information such as the number of clusters (and thus likely sources) at a given time and the potential connecting clusters might also be computed automatically for use by the interpretation process. The example will be examined from batch mode, i.e., the system has access to all of the data and must decide what data to focus on and interpret. Real-time interpretation would proceed in a similar fashion except that the system would have less flexibility in choosing from existing data and skipping around in time, but more flexibility from its ability to actively direct the evidence gathering process. The example will only make use of the given acoustic sensor data in order to form its interpretation, however, methods for making use of additional sources of evidence will also be discussed.

As described in section 5.2.8, in our approach potential control decisions are accomplished through the refinement of the system goals into hierarchical control plans. These plans represent the decisions which must be made in order to identify the evidence gathering (domain) actions to be taken. The advantage of the hierarchical refinement of the plans is that heuristic focusing knowledge can be applied at each level in the hierarchy to explicitly reason about the best control approaches to expand and examine. This is in contrast with the Control BlackBoard approach in which all decisions are made implicitly through the rating of actions (both control and domain). And-or graphs can be used to represent the two aspects of control planning: hierarchical refinement and elaboration over time. Planning operators are defined for each type of plan which can be used to refine the plan or to elaborate it over time when appropriate. Plan elaboration is usually limited to the next step since the outcome of a step may affect the choice of later actions or even eliminate these actions (in the case of failure of an earlier action).

The general goal form of this problem is: find all (and only those actual) answers. This goal is made specific in terms of a set of answer and evidence contraints. The answer contraints specify the connection between domain hypotheses and goal answers–i.e., they define which domain hypotheses represent answer evidence. In this example, track hypotheses represent answer evidence. In other situations, answers may be required to be higher-level plan instantiations representing the purpose of vehicle movements or may be limited to particular subsets of tracks such as those for vehicles which pose a threat. Evidence contraints specify acceptable evidence for goal satisfaction. Here they state that for all space-time of interest, we want to be sufficiently sure that the space is covered with an answer or with a non-answer (to be sure *all* answers are found). Just what is meant by "sufficiently sure" is specified in terms of acceptable residual sources of uncertainty in the answer evidence.

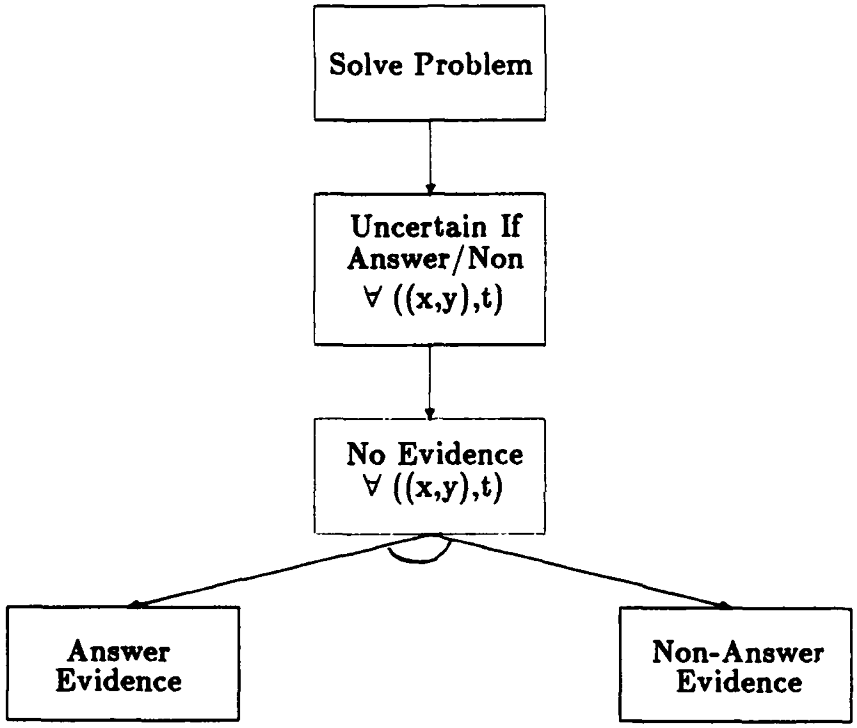


Figure 5.2: Initial refinement of problem solving goals.

The refinement of the system goal is accomplished by examining the current situation to determine the uncertainties in the goal which must be further resolved in order to meet the evidence constraints. This process is represented in figure 5.2. Initially, the system goal is unsatisfied because there is too much uncertainty over the possibility of additional answers (here, actually uncertain if *any* answers). Thus, the system posts a subplan to resolve this uncertainty. This subplan is further refined by determining the source of the uncertainty. At this point, the uncertainty exists because there is *no* evidence. A subplan is posted which directs the system to gather evidence to resolve this (no evidence) source of uncertainty. Refining this subplan involves determining potential types of evidence that can resolve this source of uncertainty. There are two basic types of evidence that can be used to resolve this uncertainty: evidence of additional (potential) answers and evidence to exclude answers (negative evidence). Subplans representing these evidence options are posted, effectively representing the possible strategies for meeting the system goals. Further refinement, then, involves the selection of appropriate actions to take to gather this goal evidence by gathering evidence for domain hypotheses.

At any level in the refinement process, there may be heuristic focusing knowledge which is applicable. For example, it may be that there is a heuristic that says to prefer negative evidence to potential answer evidence. This is reasonable since negative evidence can limit the work that the system must do by eliminating a portion of the answer space from further

consideration. Also actions that can gather negative evidence generally also can produce potential answer evidence. If there was such a heuristic, it could be used at this point to focus the planning process by limiting the subplans which are initially refined. Of course most heuristics will probably require more specific information about the characteristics of the available data and so would not be applicable at this level in the planning process. Instead, additional information would be required from further plan refinement before decisions could be made.

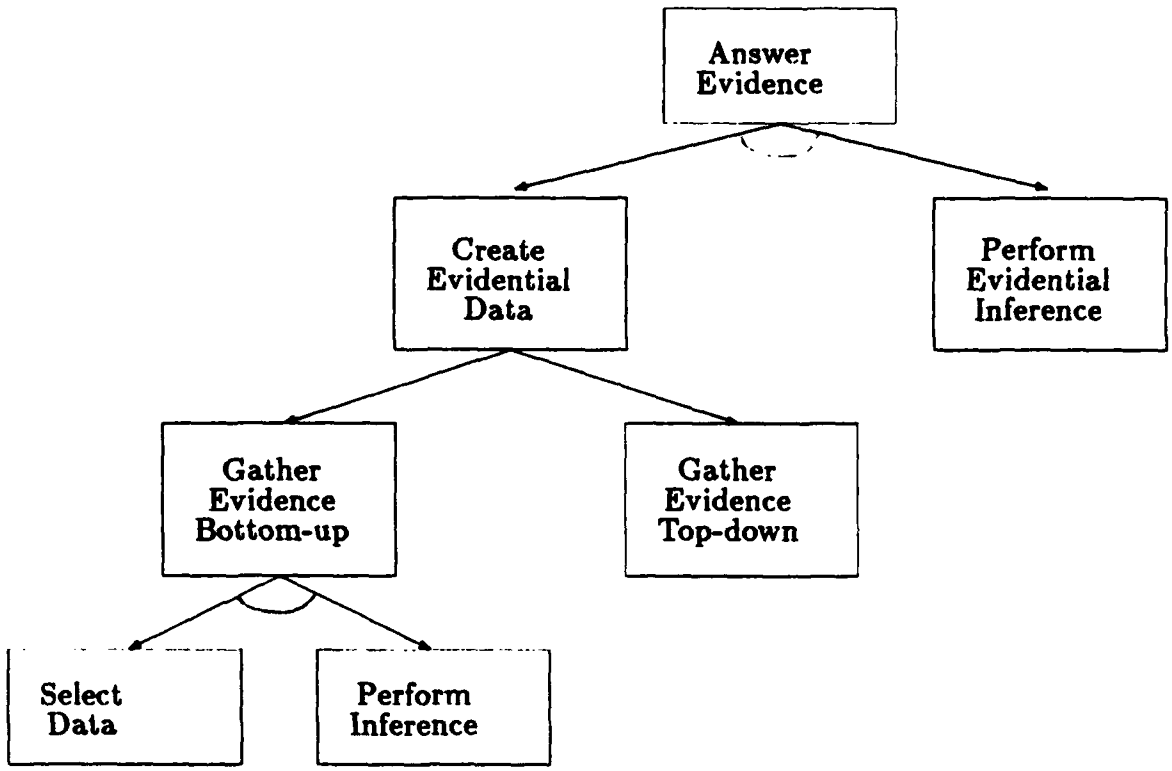


Figure 5.3: Partial plan refinement to create answer evidence.

Further refinement of the control plan now involves determining how to create the desired answer or non-answer evidence. For the purpose of illustrating the control process, we will concentrate on developing answer evidence at this point in the example. Answer evidence requires the creation of a track hypothesis so a subplan is created to represent this goal–see figure 5.3. Creating a hypothesis means performing an evidential inference to support the hypothesis. This in turn requires that there be evidential data to perform the inference from. Thus the refinement of the create track hypothesis plan is a multi-step plan which is represented in figure 5.3 as *and* nodes in the plan. The subplan to create evidential data is further refined in order to determine how to create the data. A major decision in this process is whether to pursue the data in and top-down or in a bottom-up fashion. At this point in the processing, heuristic focusing would select bottom-up pursuit

of the data because the goal hypothesis, the track, does not contain any information that would help to focus a top-down approach.

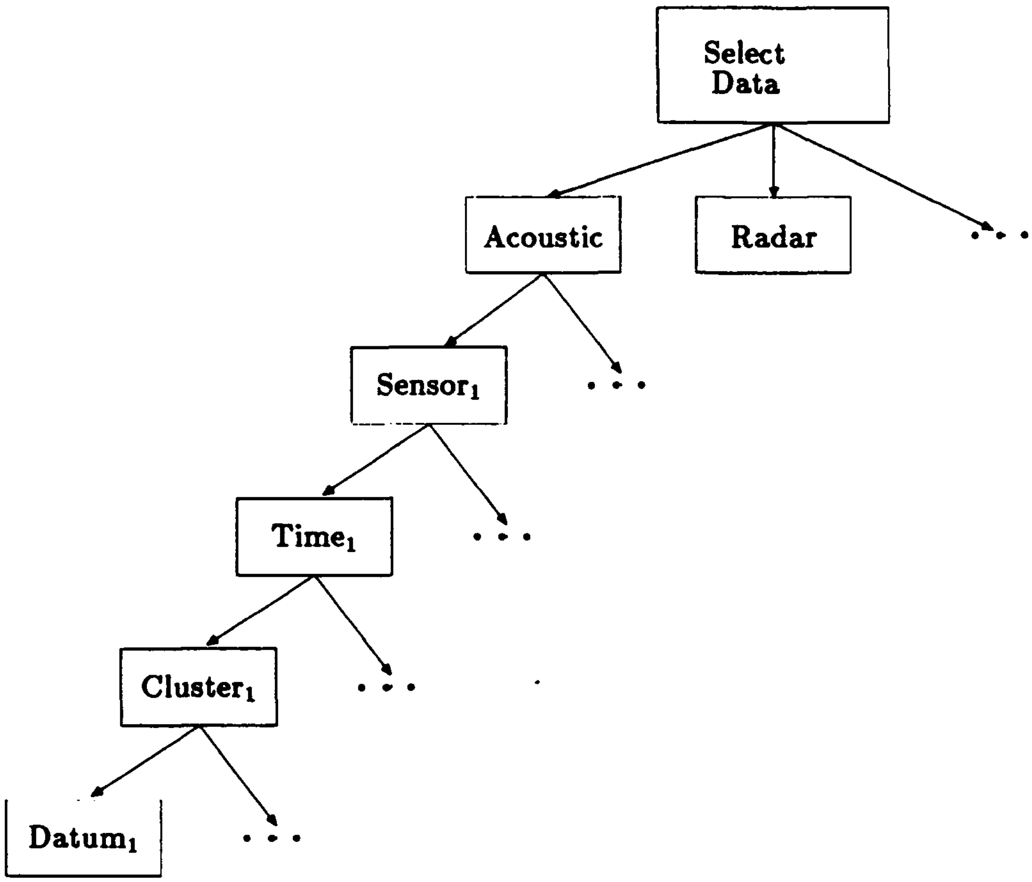


Figure 5.4: Partial plan refinement to select data.

Working bottom-up, the control process must first select or create evidential data to be used in creating evidence via plan abstraction. Concentrating on the only data we are assuming is available, the acoustic sensor data, refinement to select data involves indexing into the data through a number of stages–see figure 5.4. These refinement levels depend on how the data characteristics can be accessed and which are useful for making focusing decisions. For example, the data sets which contain the fewest number of clusters (and hence potential sources) are ranked more highly for producing evidence of potential answers since there are less possibilities to be resolved. This results in the selection of the clusters at either 3a or 4a. On the other hand, if the goal were to resolve uncertainty in a vehicle position or type, then data sets containing clusters with well sensed (loud) signals might be preferred. The reasoning here being that louder signals tend to be more accurately sensed, but are not inherently more likely to represent sources of interest–consider, for instance, high-flying aircraft or battlefield conditions. In the existing POISE focusing framework

the heuristic knowledge here would have to be represented as two rules: prefer data at times with fewer clusters and prefer better sensed (louder) signals. These heuristic rules would then conflict and POISE had no methods for resolving the conflict. By indexing our knowledge to the purpose of the actions, this conflict is avoided in the new framework. The cluster density may be specified as the primary metric when trying to develop evidence of potential answers while the signal strength is the primary metric for resolving vehicle ID and position uncertainties.

The decision sequence finally results in choosing to interpret a signal in the 3a cluster. The choice of 3a over 4a and the choice of signal in the cluster may be made randomly if there are no distinguishing characteristics (such as signal strength, etc.). The interpretation process must deal with the uncertainties in what is represented by the data. Because of this uncertain connection between data and what it represents, we choose to add an additional abstraction level to the DVMT hierarchy: the acoustic sensor data level. This level represents the data as received from the sensor and as such contains no uncertainties–the data is whatever it is. What is uncertain is just what the data represents about the environment. Each data point may represent a single environmental signal, multiple (closely spaced) signals, or no signals at all (noise/sensor malfunction). Likewise, we are uncertain about the actual frequency and position of signals represented by the data points because of the resolution limitations of the sensor.

Once data has been selected, the select data plan is complete. This causes the plan to be updated (by the updating operator associated with the plan) so that the infer evidence subplan is now active. Since there may be a number of ways to interpret data, the interpretation process must be guided by heuristic control knowledge. Thus, the plan to infer evidence must be further refined before action can be taken. Decisions must be made about which whch uncertainties to pass along, how to represent these uncertainties, and what hypotheses to create. As always, focusing heuristics are able to reference the control context in order to determine the most appropriate decisions for the situtation. For example, because of the lack of (existing) evidence regarding proper frequency and position and the fact that the signal strength was very low, it makes sense to reserve judgment and pass along these uncertainties. Thus the result of the interpretation process is a signal level hypothesis which is uncertain in frequency and position.

The result of the evidential inference is the creation of a signal-level hypothesis connected by an evidence link to the supporting data. This is the lowest level link in figure 5.5 The hypothesis is uncertain–that is, it is uncertain whether the signal hypothesis represents an actual environmental signal–because the data may be the result of sensor malfunction. This uncertainty is represented by attaching appropriate symbolic tags to the evidence link which stand for the sources of uncertainty in the evidence (these are not shown in figure 5.5). Acoustic sensor data support for a signal hypothesis includes the sources of

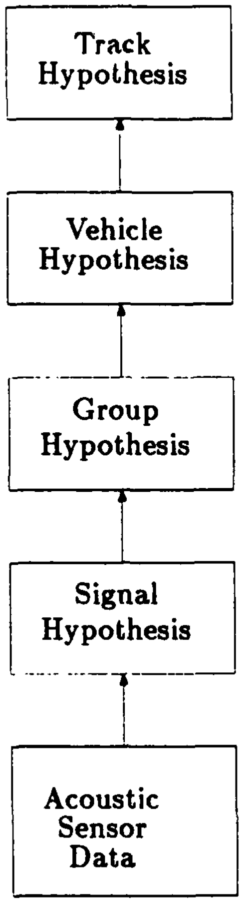


Figure 5.5: Evidential support for track hypothesis.

uncertainty: sensor-noise and sensor-ghost. Uncertainty in the parameter values of the signal hypothesis is represented by range values for the parameters. For example, the sensor resolution characteristics tell us that the actual frequency of the environmental signal is within one frequency class of the sensor data so the signal hypothesis frequency is $f \pm 1$ where f is the acoutic data signal frequency class.

The creation of a signal hypothesis satisfies the evidence creation step of the bottom-up evidence gathering plan. This causes the plan to be updated so that the newly created hypothesis can be used to pursue the desired track evidence–see figure 5.6. Of course, in a real-time vehicle monitoring system other sources of evidence would typically have become available in the intervening period and would also have to be evaluated relative to the new hypothesis. For example, if radar data required active accumulation there would be a delay between the data request and the availability of the data. Thus it would be possible for radar data requested before the last cycle to be available at this point and be preferred to the just created signal data.

Updating the bottom-up evidence gathering plan produces two new subplans: one to

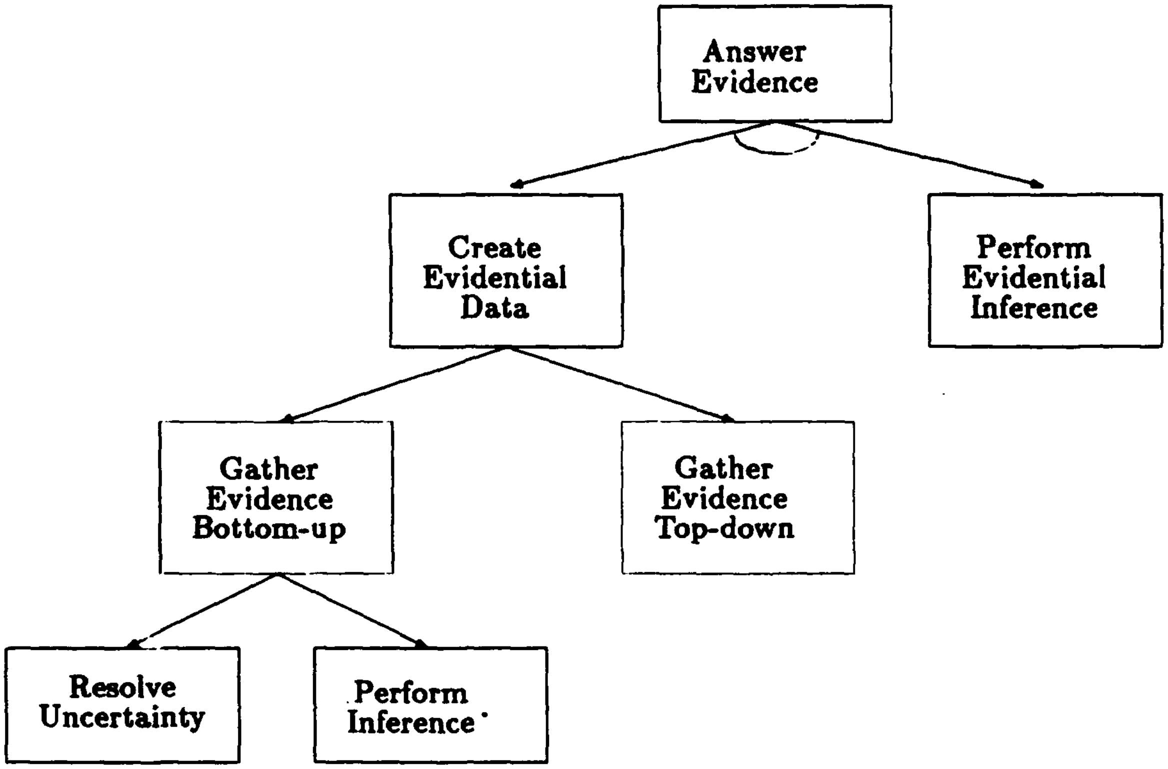


Figure 5.6: Partial plan refinement following hypothesis creation.

create further evidence from the newly create hypothesis and one to resolve uncertainty in that hypothesis. The signal hypothesis is uncertain due to the uncertainty in its supporting evidence. Because of the explicit recording of evidence and its associated sources of uncertainty, the control process can consider exactly why hypotheses are uncertain and what actions are appropriate to resolve the uncertainty. This is the sense in which the sources of uncertainty information is used to drive the control process. The sources of the uncertainty in the signal hypothesis are the potential (alternative interpretations) of the acoustic sensor data as sensor noise or sensor ghosting. Thus planning how to resolve uncertainty in the correctness of the hypothesis requires gathering evidence to resolve these sources of uncertainty–if actions are available to accomplish this. For example, we could imagine an action of running diagnostics on the sensor to determine the possibility that the sensor is malfunctioning.

On the other hand, the focusing heuristics may suggest that it is more appropriate to pursue this hypothesis further before accumulating additional evidence. The result of interpreting a signal hypothesis is the creation of a group hypothesis. Creation of a group hypothesis causes the bottom-up evidence gathering plan to be updated in much the same way as after the creation of the signal hypothesis. In particular, there is uncertainty over whether the signal is part of a group or not (it may simply be a solitary signal), over the group class because of the frequency uncertainty in the signal, and over the position of

the group because of the position uncertainty in the signal. Because additional sources of evidence for the group exist, heuristic focusing knowledge may suggest that accumulating additional evidence for the group hypothesis is preferable to further abstraction. The sources of uncertainty in the group hypothesis are from incomplete signal evidence (no evidence of the other signals making up the group), uncertainty over the support of the existing signal hypothesis (it may not be part of a group), and the uncertainty in the supporting signal hypothesis itself. The incomplete signal evidence uncertainty can be resolved by attempting to generate additional, appropriate signal hypotheses. Potential acoustic data to abstract is identified through the cluster relation with the already abstracted datum.

Eventually, a vehicle hypothesis will be created from this group and then a track hypothesis will be created from the vehicle hypothesis. The resulting levels of evidential support are represented in figure 5.5. Explicit links are maintained between representations of data or hypotheses and the hypotheses this evidence supports. Associated with each evidential link is information about the type of the inference (i.e., the role the evidence plays in support of the hypothesis) and the sources of uncertainty in the inference. Since uncertainty in the track hypothesis results from uncertainty in the evidence supporting the track, the sources of uncertainty information associated with the evidential links explicitly represents the sources of uncertainty in the track hypothesis. Thus, "proving" the track hypothesis correct means planning how to resolve the uncertainty in the track hypothesis by identifying actions which can resolve these sources of uncertainty. The sources of uncertainty in the track evidence result from the incomplete set of vehicle hypotheses supporting the track and from the uncertainty in the vehicle hypothesis due to the sources of uncertainty in its supporting evidence.

The track hypothesis satisfies part of the answer goal in that it is of the correct plan type to be an answer. This resolves the problem solving goal uncertainty as to whether the acoustic signal evidence is capable of supporting a valid answer or not. However, additional evidence must still be developed to satisfy the evidence constraints by sufficiently "proving" the hypothesis and by sufficiently refining hypothesis' parameters such as position and vehicle type. The high-level portion of the control plan as it exists following the creation of a potential answer hypothesis is shown in figure 5.7. The partial vehicle evidence uncertainty is resolved by generating additional vehicle hypotheses to support the track–i.e., tracking the vehicle. The planning process makes use of the plan constraints to refine the plans in order to limit the actions and data which are deemed relevant. In tracking, constraints can be used to limit the data to be examined based on frequency class and position.

At this point, the track could be extended using time 2 data or time 4 data. Since only a single cluster is potentially applicable at time 4 the focusing heuristics would probably

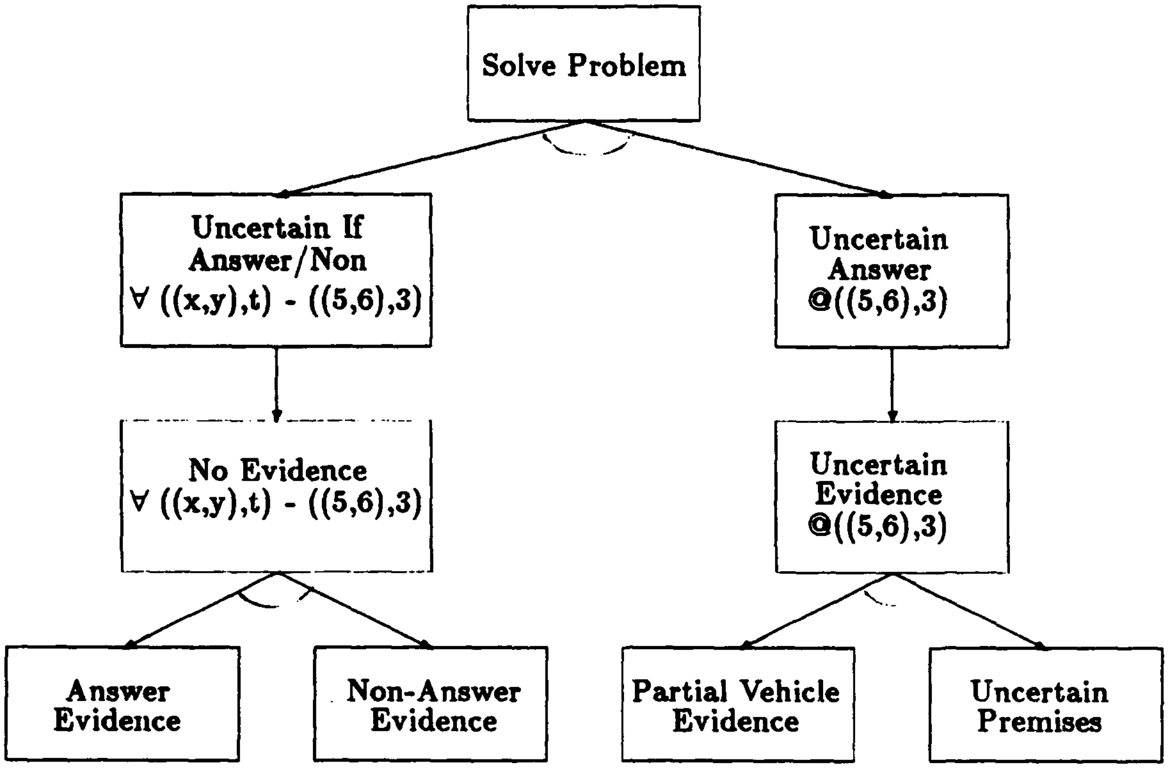


Figure 5.7: Partial plan refinement once potential answer created.

select this as the data to use for the next extension of the track. This process would continue until the track is deemed to have sufficient supporting evidence to meet the evidential constraints. This may or may not require the construction of a "complete" track. If it does not and a complete track is required of valid answers then tracking would continue–driven by the valid answer problem solving goal. In this example, tracking is quite straightforward because the focusing process directed the crucial time 3 and time 4 data to be used to construct the basis of the track. This segment is incompatible with the "b" track extensions because of kinematic constraints. Thus, there is never the possibility of multiple, alternative extensions here. In general, however, there will be alternative extensions for hypotheses. Alternatives are identified by "alternative links" set up between the extension hypotheses. Alterntive relations represent a general type of negative evidential relationship which can exist between any two hypotheses. That is, evidence for one of the alternatives is considered as evidence against the alternative and vice versa. The addition of this negative evidence causes conflict uncertainty which must be resolved by resolving uncertainty in the alternative extensions.

Even when only acoustic sensor data is used to form the interpretations, it need not always be used in the same way. For example, as additional evidence is gathered for the track, confidence in the correctness of the track and the vehicle type increases. At some point, it may become reasonable to generate less certain, but also less expensive evidence by

abstracting directly from a data cluster to a vehicle hypothesis (using a different evidence-generating KS). This evidence would contain some rather significant sources of uncertainty were it to be considered as a solitary inference because the consistency of the frequency information would not have been established. However, in the overall track hypothesis these sources of uncertainty may be of little consequence and are offset by the advantages of faster processing. This same sort of option occurs with other sources of evidence such as radar. Resolution can be controlled by varying the power and scan speed with the tradeoffs being the risk of missing certain targets and limited coverage area.

To fully meet the system goal of creating all and only the correct hypotheses, the system must also investigate space-time not covered by the "a" track in order to develop evidence that there are no additional answers. Unlike real acoustic sensor data, there is little "noise" in this simple example. In general, acoustic sensor data would probably not be the best possible source of negative answer evidence because it tends to be relatively noisy. To produce negative evidence, acoustic sensor data is interpreted differently than when producing potential answer evidence. An entire time slice is interpreted at once with the absence of any data clusters producing direct negative evidence for vehicles and clusters producing direct (but very uncertain) evidence of vehicles. Any vehicle hypotheses produced would be a source of uncertainty in the system goals since they would fail to meet the evidential constraints. The system must resolve these uncertainties by gathering additional evidence to determine whether this evidence represents an answer or not.

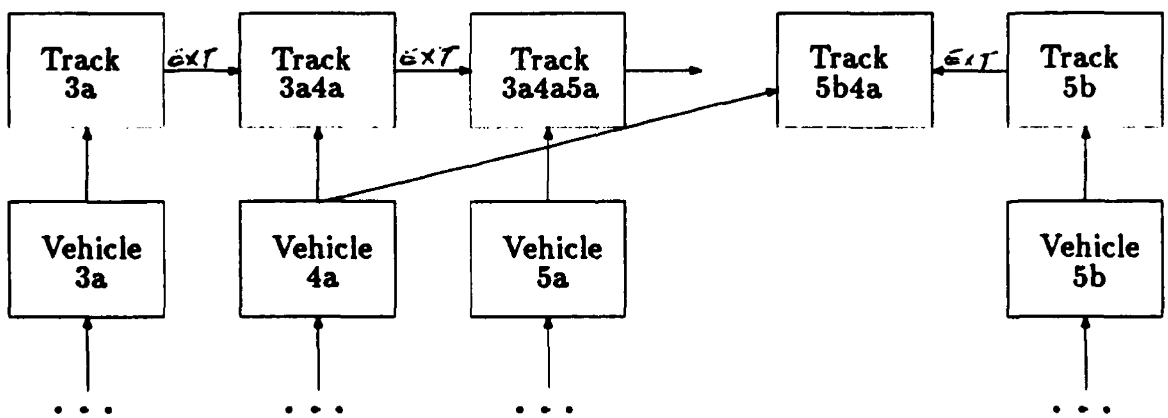


Figure 5.8: Alternative tracks due to shared vehicle hypothesis.

In the example, this means that the system would have to examine the "b" track data to produce sufficient evidence that this data does not support an answer track. The exact negative evidence generated depends upon the order in which the system pursues the alternative interpretations of this data. What is clear is that any tracks involving "b" data would have to include "a" data from times 3 and 4 in order to be extended

and completed. However, attempting to extend a "b" track into these areas causes the system to recognize the "b" and "a" tracks as alternatives because they share vehicle evidence. This results in an alternatives evidential relationship being set up between the "a" and "b" track hypotheses. Figure 5.8 shows such a conflict resulting from one potential track construction scenario. The addition of this conflicting negative evidence to the tracks results in an additional source of uncertainty which must be resolved by resolving the other sources of uncertainty in the hypotheses. Since the "a" track is well supported, this conflict causes little additional uncertainty and so it is probably best to resolve the uncertainty in the "b" tracks by trying to extend them. However, there are no possible track extensions for the "b" tracks here because the constraints on the vehicle kinematics prohibit these tracks from containing both 3a and 4a data. This extension failure is an additional source of strong negative evidence for the "b" tracks. Of course, this negative evidence is still somewhat uncertain since the possibility of missing data at times 3 and 4 is a source of uncertainty for extension failure evidence. Should the combination of negative evidence from the "a" track and from the extension failure not be deemed sufficient proof against the "b" tracks then additional actions would be needed to try to resolve the uncertainty. These actions would involve pursuing the sources of uncertainty in both the positive and negative evidence for the "b" segments. For example, by gathering additional evidence for the "a" track, resolving whether the "b" segments fit the criteria for sensor ghosts of the "a" track, and postulating missing data.

The control reasoning in this example can be more involved than in the standard DVMT since it can be made to rely heavily on domain knowledge about evidence and uncertainty. This is exactly the point of this work: basing control on a process of accumulating explicit, symbolic evidence to manage and resolve uncertainties makes it possible to reason in more detail about control decisions. In this example, heuristic control knowledge greatly limits the amount of work that is done by focusing the problem solving process on the data which is the most promising for meeting the goals. The DVMT spends a great deal of effort building and re-building track segments for the "b" data without recognizing the crucial role of data at times 3 and 4 and without recognizing the redundancy of its actions (this problem has been addressed in a different manner in recent work [22]). Our approach also provides a better basis for understanding the solution. This is a particularly problematic example for the DVMT since the "solution" track data is weaker than the "ghost" track data. In the DVMT, constraints on vehicle kinematics are used to eliminate the ghost track from consideration. Since we wish to consider the possibility of mis-sensed and missing data the problem becomes more difficult. Postulating missing data at time 3 and/or 4, it is possible to complete the "ghost" track. While this alternative could presumably be eliminated from consideration by "tuning" the procedure for weighing evidence, in real problems there would be other sources of evidence to resolve this uncertainty. For

example, it seems extremely unlikely for there to be *no* sign of the actual vehicle at these times and complete signal data for the incorrect track. These would be considered as strong sources of evidence in resolving the missing data track extension alternative. In any case, by maintaining an explicit record of evidence, uncertainties, and assumptions, our system provides a basis for understanding the remaining sources of uncertainty in the interpretation.

Chapter 6

Conclusion

This document describes a plan recognition framework which addresses the limitations of existing plan recognition systems. A plan recognition system of any sophistication and generality must be able to meet the requirements laid out in section 4.4. Our approach seems to have all of these qualities. However, evidence-based plan recognition requires the development of techniques which are of more general AI interest as well. The key characteristics of our approach include:

- Evidence and sources of uncertainty are explicitly represented.
- Heuristic control decisions are based on the sources of uncertainty in the hypotheses and the need to manage uncertainty.

These techniques could be exploited by many systems if their use was better understood. A system must have access to the reasons for its beliefs if it is to be able to reason intelligently about control decisions. An explicit representation of the sources of evidence makes it possible to understand the *sources of uncertainty* in the beliefs. Evidence provides uncertain support for a hypothesis because there are conditions under which the evidence may fail to support the hypothesis. Numeric rating functions gathered from experts typically summarize just such knowledge–along with a priori likelihood judgements. Explicit information about the uncertainties in evidence is a type of knowledge that we feel is very important for the development of more sophisticated AI systems. It allows us to evaluate belief dynamically rather than having to rely on a priori likelihoods since it is now possible to enumerate the sources of uncertainty in evidence and judge their likelihood in the current contexts. Control decisions can be directed toward gathering the best evidence to resolve the most critical sources of uncertainty. That is, we can *manage* uncertainty rather than just trying to *resolve* it because we understand exactly what the sources of that uncertainty are, which are most critical, and what evidence is best. This applies whether or not the system can interact with its environment to affect the evidence it has available.

Bibliography

[1] Adams, J. Barclay, "Probabilistic Reasoning and Certainty Factors," in Bruce Buchanan and Edward Shortliffe, editors, *Rule-Based Expert Systems*, Addison-Wesley, Reading, 1984, 263–271.

[2] Allen, James, "Recognzing Intentions From Natural Language Utterances," from *Computational Models of Discourse*, Michael Brady and Robert C. Berwick, editors, The MIT Press, 1983, 107–164.

[3] Azarewicz, Jerome, Glenn Fala, Ralph Fink, and Christof Heithecker, "Plan Recognition for Airborne Tactical Decision Making," submitted to AAAI-86, 1986.

[4] Barton, George, Jr., *A Multiple-Context Equality-Based Reasoning System*, Technical Report AI-TR-715, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1983.

[5] Buchanan, Bruce and Edward Shortliffe, editors, *Rule-Based Expert Systems*, Addison-Wesley, 1984.

[6] Carver, Norman, Victor Lesser, and Daniel McCue, "Focusing in Plan Recognition," *Proceedings of AAAI-84*, 1984, 42–48.

[7] Charniak, Eugene and Drew McDermott, *Introduction to Artificial Intelligence*, Addison-Wesley, 1985.

[8] Clancey, Willian, "Extensions to Rules for Explanation and Tutoring," in Bruce Buchanan and Edward Shortliffe, editors, *Rule-Based Expert Systems*, Addison-Wesley, Reading, 1984, 531–568.

[9] Cohen, Paul and Milton Grinberg, "A Theory of Heuristic Reasoning About Uncertainty," *The AI Magazine*, Summer 1983, 17–24.

[10] Cohen, Paul, *Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach*, Pitman, 1985.

[11] Cohen, Paul, David Day, Jeff Delisio, Mike Greenberg, Rick Kjeldson, Dan Suthers, and Paul Berman, "Management of Uncertainty in Medicine," *International Journal of Approximate Reasoning*, Spring 1987.

[12] Corkill, Daniel, *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*, Ph.D. Thesis and Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, 1982.

[13] Davis, Randall, "Meta-Rules: Reasoning about Control," *Artificial Intelligence,* 15 (1980), 179–222.

[14] Davis, Randall and Douglas Lenat, *Knowledge-Based Systems in Artificial Intelligence,* McGraw-Hill, New York, 1982.

[15] Davis, Randall and B. Buchanan, "Meta-Level Knowledge: Overview and Applications," in Bruce Buchanan and Edward Shortliffe, editors, *Rule-Based Expert Systems*, Addison-Wesley, Reading, 1984, 507–530.

[16] deKleer, Johann, "Choices Without Backtracking," *Proceedings of AAAI-84,* 1984, 79–85.

[17] Doyle, Jon, "A Truth Maintenance System," *Artificial Intelligence,* 12 (1979), 231–272.

[18] Doyle, Jon, *A Model for Deliberation, Action, and Introspection,* Technical Report AI-TR-581, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1980.

[19] Doyle, Jon and Philip London, "A Selected Descriptor-Indexed Bibliography to the Literature on Belief Revision," *SIGART Newsletter,* No. 71, 1980, 7–23.

[20] Doyle, Jon, *Some Theories of Reasoned Assumptions,* Technical Report CMU-CS-83-125, Department of Computer Science, Carnegie-Mellon University, 1983.

[21] Doyle, Jon, "Methodological Simplicity in Expert System Construction: The Case of Judgements and Reasoned Assumptions," *The AI Magazine,* Summer 1983, 39–43.

[22] Durfee, Edmund and Victor Lesser, "Incremental Planning to Control a Blackboard-Based Problem Solver," submitted to *AAAI-86,* 1986.

[23] Garvey, Alan, Craig Cornelius, and Barbara Hayes-Roth, "Computational Costs versus Benefits of Control Reasoning," *Proceedings of AAAI-87,* 1987, 110–115.

[24] Green, Peter, "Distributed Acoutic Surveillance and Tracking," *Proceedings of Workshop on Distributed Sensor Networks,* Lincoln Laboratory, Massachusetts Institute of Technology, 1982.

[25] Gordon, Jean and Edward Shortliffe, "The Dempster-Shafer Theory of EVidence," in Bruce Buchanan and Edward Shortliffe, editors, *Rule-Based Expert Systems*, Addison-Wesley, Reading, 1984, 272–292.

[26] Hayes-Roth, Barbara, "A Blackboard Architecture for Control," *Artificial Intelligence*, 26, 1985, 251–321.

[27] Hayes-Roth, Barbara, et. al., "PROTEAN: Deriving Protein Structure from Constraints," *Proceedings of AAAI-86*, 1986, 904–909.

[28] Hayes-Roth, Barbara, et. al., *A Layered Environment for Reasoning about Action*, Technical Report KSL 86-38, Knowledge Systems Laboratory, Computer Science Department, Stanford University, 1987.

[29] Hayes-Roth, F. and R. Wesson, "Distributed Intelligence for Situation Assessment," *Proceedings of Workshop on Distributed Sensor Nets*, Carnegie-Mellon University, 1978.

[30] Hudlicka, Eva and Victor Lesser, "Meta-Level Control Through Fault Detection and Diagnosis," *Proceedings of AAAI-84*, 1984, 153–161.

[31] Huff, Karen, and Victor Lesser, *Knowledge-Based Command Understanding*, Technical Report 82-6, Department of Computer and Information Science, University of Massachusetts, 1982.

[32] Johnson, M. Vaughn, Jr. and Barbara Hayes-Roth, "Integrating Diverse Reasoning Methods in the BB1 Blackboard Control Architecture," *Proceedings of AAAI-87*, 1987, 30–35.

[33] Kautz, Henry and James Allen, "Generalized Plan Recognition," *Proceedings of AAAI-86*, 1986, 32–37.

[34] Kautz, Henry, *A Formal Theory of Plan Recognition*, Technical Report TR 215, Department of Computer Science, University of Rochester, 1987.

[35] Lacoss, R., "Sensor Tutorial," *Proceedings of Workshop on Distributed Sensor Nets*, Carnegie-Mellon University, 1978.

[36] Lenat, Douglas, Randall Davis, Jon Doyle, Michael Genesereth, Ira Goldstein, and Howard Schrobe, "Reasoning about Reasoning," in F. Hayes-Roth, D. Waterman, and D. Lenat, editors, *Building Expert Systems*, Addison-Wesley, Reading, MA, 1983.

[37] Lesser, Victor and Daniel Corkill, "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks," *The AI Magazine*, Fall, 1983, 15–33.

[38] Litman, Diane and James Allen, *A Plan Recognition Model for Subdialogues in Conversations*, Technical Report TR-141, Department of Computer Science, University of Rochester, 1984.

[39] London, Philip, *Dependency Networks as a Representation for Modeling in General Problem Solvers*, Technical Report TR-698, Department of Computer Science, University of Maryland, 1978.

[40] Lowrance, John and Thomas Garvey, *Evidential Reasoning: An Implementation for Multisensor Integration*, Technical Note 307, SRI International, 1983.

[41] McCallester, David, *The Use of Equality in Deduction and Knowledge Representation*, Technical Report AI-TR-550, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1980.

[42] McCallester, David, *An Outlook on Truth Maintenance*, AI Memo No. 551, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1980.

[43] Quine, W.V. and J.S. Ullian, *The Web of Belief*, Random House: New York, 1970.

[44] Rich, Elaine, *Artificial Intelligence*, McGraw-Hill, New York, 1983.

[45] Schmidt, Charles, and N.S. Sridharan, "Plan Recognition using a Hypothesize and Revise Paradigm," *Proceedings of IJCAI-77*, 1977, 480–486.

[46] Schmidt, C.F., N.S. Sridharan, and J.L. Goodson, "The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence," *Artificial Intelligence*, 11 (1978), 45–83.

[47] Shortliffe, Edward and Bruce Buchanan, "A Model of Inexact Reasoning in Medicine," in Bruce Buchanan and Edward Shortliffe, editors, *Rule-Based Expert Systems*, Addison-Wesley, Reading, 1984, 233–262.

[48] Smith, Reid, Howard Winston, Tom Mitchell, and Bruce Buchanan, *Representation and Use of Explicit Justifications*, Knowledge Systems Laboratory Working Paper No. KSL 85-3, Computer Science Department, Stanford University, 1985.

[49] "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence*, 16, 1981, 141-170.

[50] Sullivan, Michael and Paul Cohen, "An Endorsement-Based Plan Recognition Program," *Proceedings of IJCAI-85*, 1985, 475–479.

[51] Thompson, Alan, "Network Truth-Maintenance for Deduction and Modelling," *Proceedings of IJCAI-79*, 1979, 877–879.

[52] Winston, Patrick, *Artificial Intelligence*, second edition, Addison-Wesley, Reading, MA, 1984.

[53] Wong, Douglas, "Language Comprehension in a Problem Solver," *Proceedings of IJCAI-81*, 1981, 7–12.

[54] Wilensky, Robert, *Planning and Understanding* Addison-Wesley, Reading, MA, 1983.

Knowledge Acquisition through Anticipation of Modifications

A Dissertation Presented

by

Lawrence S. Lefkowitz

Submitted to the Graduate School of the University of Massachusetts in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy

September 1987

Computer and Information Science

KNOWLEDGE ACQUISITION THROUGH
ANTICIPATION OF MODIFICATIONS

A Dissertation Presented

by

LAWRENCE S. LEFKOWITZ

Approved as to style and content by:

Victor R. Lesser, Chair of Committee

W. Bruce Croft, Member

Paul R. Cohen, Member

Klaus Schultz, Outside Member

W. Richards Adrion, Department Chair
Computer and Information Science

In memory of

Leontine Lefkowitz

ACKNOWLEDGMENTS

I would like to thank the members of my dissertation committee for their support, insights, perseverance and patience. Especially, I would like to thank Victor Lesser for allowing me the freedom to explore my ideas (and providing an environment in which I could do so), for challenging those beliefs I had mistaken for truths, for reminding me to occasionally examine my results, and for forcing me to see the big picture when my field of view became too narrow. Victor has a unique talent for viewing problems from unusual perspectives. The directions in which he has steered this research, though occasionally not clear to me at the time, were almost always productive.

When I would lose sight of what needed to be done and would be teetering on the brink of an intellectual bottomless pit, Bruce Croft was usually the one responsible for getting me back on track. His organizational ability and clear sense of direction are largely responsible for this dissertation having been completed. My discussions with Paul Cohen enabled me to make this work more complete in another sense; his familiarity with the relevant literature helped me to place this work in its appropriate context. Much of the psychological work on knowledge assimilation towards which he pointed me proved quite valuable. Finally, as someone less enmeshed in the world of computer science, Klaus Schultz prevented me from assuming that what was clear to me would be clear to everyone else. When I failed to present things in an understandable fashion, Klaus kept me honest by pointing out some of the gaps I had flown over.

This work draws heavily upon ideas originally developed in the POISE project. Many of these ideas arose from work done in cooperation with Norman Carver and Carol Broverman. I would like to thank them for their contributions to the concepts, architecture and implementation of that system and for the frequent and prolonged discussions, intellectual and otherwise, that transpired along the way.

The Research Computing Facility has provided the hardware and software support that has made the implementation of the $\mathrm{K^{n}{}_{A}c}$ system possible. I would like to express particular thanks to Art Gaylord and Gary Wallace who were never too busy to come to my aid and convince some of the more stubborn bits to go where I believed they should be.

On a financial note, I would again like to thank Victor Lesser and Bruce Croft for assuring that I always had a roof over my head during my long "career" as a graduate student. Thanks also to the Digital Equipment Corporation for their support of the POISE project and to the Northeast Artificial Intelligence Consortium and the Rome Air Development Center for funding the $\mathrm{K^{n}{}_{A}c}$ research.

This dissertation would not have been possible were it not for the personal and emotional support of my family throughout this endeavor. I would like to thank my parents, Lou and Lee Lefkowitz, and my sister, Leslie Turken, for their faith in me and for their support whenever I most needed it. I must also thank my second family, Al Hough, Lee Thomson, and Bill and Cathy Verts, whose friendship, warmth, wit and tolerance of bad puns helped me to keep things in perspective.

Finally, I would like to thank my wife, Judy Snape, for believing that this

dissertation would someday be completed, despite much evidence to the contrary; for putting up with nights I would not come to bed and mornings I would not leave it; for standing by me through the bad times and for providing so many of the good times; and for always being there when I needed a friend.

ABSTRACT

KNOWLEDGE ACQUISITION THROUGH ANTICIPATION OF MODIFICATIONS

September 1987

Lawrence S. Lefkowitz, B.E., Cooper Union
M.S., Ph.D., University of Massachusetts

Directed by: Professor Victor R. Lesser

Knowledge bases for expert systems are typically constructed and refined using information obtained through a series of dialogs between an expert in the application domain and a knowledge engineer. The assimilation of this information into an existing knowledge base is an important aspect of the knowledge engineer's task. This assimilation process requires an understanding of how the new information corresponds to that already contained in the knowledge base and how the existing information must be modified so as to reflect the expert's view of the domain.

This work describes a new approach to knowledge acquisition and presents a system, K^{n}Ac, which implements this approach. K^{n}Ac modifies an existing knowledge base using information obtained during a discourse with a domain expert. Heuristic knowledge about the knowledge acquisition process enables K^{n}Ac to anticipate modifications to existing entity descriptions. These anticipated modifications, or *expectations*, are used to provide a context in which to assimilate the new domain information.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

The desire of knowledge, like the thirst of riches,
increases ever with the acquisition of it.
— Laurence Sterne, *Tristram Shandy* (1760)

As artificial intelligence systems move out of the laboratory and attempt to confront real-world applications, the need for large, potentially complex knowledge bases becomes clear. The creation and maintenance of these knowledge bases has proven to be a severe bottleneck in the development of these "intelligent" systems. Identifying the necessary knowledge, both domain-specific and more general "world" knowledge, providing a representation capable of describing its salient features, extracting this information from the appropriate "domain experts", and assimilating this information into an existing knowledge base have each proven to be a formidable task. This work addresses these issues and presents a system, $\mathrm{K^{n}{}_{A}c}$, which offers solutions to the knowledge assimilation aspect of this knowledge acquisition problem.

As the process of transferring knowledge from a domain expert into an expert system's knowledge base becomes a more substantial portion of the system's total development cost, several approaches are being examined to automate this labor-

intensive task. These approaches either provide the knowledge engineer with better tools, provide an automated intermediary between the domain expert and the knowledge base, or allow the expert to access the knowledge base directly. The lines between these approaches is not always distinct; as knowledge base tools become more sophisticated, they may better approximate the role played by the knowledge engineer.

The desire to make the knowledge base more accessible to the domain expert has resulted in more perspicuous knowledge representations, improved tools for editing and viewing knowledge structures, checks for consistency and completeness, etc. While these are certainly desirable steps, they focus on the *tools* for knowledge acquisition, often with little regard for the *process* involved. In this dissertation, the expertise required to perform the knowledge acquisition process is examined and a testbed for refining this expertise is presented.

§1. Knowledge Acquisition as Knowledge Assimilation

Consider the typical means by which knowledge bases are currently constructed. This usually involves a series of dialogs between an expert, or experts, in the application domain and a knowledge engineer familiar with the target expert system. The knowledge engineer's task is the modification of the expert system's knowledge base so as to reflect the domain expert's knowledge. The goal of this work is to understand and automate the knowledge engineer's ability to assimilate the information provided by the domain expert into an existing knowledge base.

To a large extent, this knowledge acquisition task may be viewed as a recogni-

tion problem. Specifically, "data" provided by the expert must be integrated into an appropriate knowledge base context. All of the problems facing other recognition systems are present here as well, including: noisy data (i.e., incomplete or inaccurate information), ambiguous interpretations, and the need to produce intermediate results before all the data is available. Thus, a significant portion of this interactive knowledge acquisition task is a matching problem: How does the expert's description of the domain correlate with the description contained in the knowledge base? How should the knowledge base be modified based on new information from the expert? What should be done when the expert's description differs from the existing one?

This matching process, while in some ways similar to that found in most recognition or interpretation systems, displays certain characteristics unique to knowledge acquisition. In particular, since the goal of a knowledge acquisition dialog is the modification of the knowledge base, the information provided by the domain expert often will not completely match the existing entity descriptions. The matching process must be modified so as to be able to recognize and, where possible, anticipate these discrepancies. $K^{n}{}_{A}c$ accomplishes this through a (modifiable) set of heuristics about various aspects of the knowledge acquisition process.

§2. The $K^{n}{}_{A}c$ System

$K^{n}{}_{A}c$ was developed to implement this knowledge assimilation approach to knowledge acquisition. It was initially designed to assist in the construction of knowledge bases for the POISE [CLLH82] intelligent interface system. These knowledge bases use a frame-like representation, described more fully in Chap-

ter 3 Section §3., to describe *tasks*, *objects* and *relationships* in the application domain. POISE's initial knowledge bases, for the office automation and software engineering domains, were created by hand from interviews between a knowledge engineer and the appropriate domain experts. Transcriptions of these interviews were examined and the results served as the basis of the K$^{n}_{A}$c system.

K$^{n}_{A}$c obtains, through a user interface, entity descriptions (e.g., tasks, objects, etc.) from the domain expert. These descriptions are compared with descriptions selected from the existing knowledge base. If any existing descriptions sufficiently match those provided by the expert, within the context of anticipated modifications, the modifications implied by any discrepancies between them are made to the existing entity descriptions. This matching process requires the ability to compare fairly complex knowledge structures, such as the event and object descriptions found in POISE, and to evaluate the results of such comparisons in the framework of a knowledge acquisition dialog. K$^{n}_{A}$c contains a structure matcher and match evaluator designed for this purpose.

The anticipation of modifications to the existing knowledge, used in the selection of potential matches for the expert's descriptions and in the evaluation of the resulting comparisons, results from K$^{n}_{A}$c's understanding of the knowledge acquisition process. This understanding is captured in the form of heuristics about several aspects of the task. Specifically, the system contains heuristics which predict modifications based on cues about the knowledge acquisition discourse, the state of the existing knowledge base, and previously made modifications. K$^{n}_{A}$c generates expectations based on these heuristics and determines which are viable at a particular time based on their certainty, the extent to which they have already been satisfied, and the passage of time and/or the change in state of the knowledge base since the expectations were generated.

§3. Outline of the Dissertation

$\mathrm{K^{n}{}_{A}c}$'s view of knowledge acquisition as knowledge assimilation is presented in the following chapter. This approach is compared with other attempts to address the knowledge acquisition problem. The assimilation process is compared with other relevant work on matching and recognition/interpretation systems.

To provide a context in which to study the knowledge acquisition process, interviews recorded during the development of an office automation knowledge base for the POISE system [CL84] were examined. Analysis of these interviews, and of the resulting modifications to the knowledge base, formed the basis of the $\mathrm{K^{n}{}_{A}c}$ system. The interviews themselves are described in Chapter 3 along with examples of the type of changes to the knowledge resulting from these interviews.

The architecture of the $\mathrm{K^{n}{}_{A}c}$ system, described briefly above, is presented in detail in Chapter 4. The system's functionality is illustrated with an example derived from the knowledge acquisition dialogs in the office automation domain.

The assimilation of the domain expert's information into the existing knowledge base requires the ability to compare knowledge structures. The process of comparing these descriptions in a manner suitable for the knowledge acquisition task and the evaluation of the resulting matches are described in greater detail in Chapter 5.

A major factor enabling $\mathrm{K^{n}{}_{A}c}$ to assimilate new information is its ability to anticipate modifications to the existing knowledge base. These anticipated modifications, or "expectations", are generated from a collection of heuristics about the knowledge acquisition process. The method by which these expectations are generated and managed is presented in Chapter 6.

As a testbed for knowledge acquisition, many facets of the $K^n{}_Ac$ system may be customized by the user. The ways in which the system can be tuned and the means for observing and evaluating the resulting changes in the system's performance are described in Chapter 7. The results of assimilating a portion of the knowledge acquisition discourse contained in Appendix A are also presented.

Finally, the contributions made by this work and the status of the $K^n{}_Ac$ system are summarized in Chapter 8. An evaluation of the system's performance is presented and directions in which this work may be extended are described.

Chapter 2

An Approach to Knowledge Acquisition

The term "knowledge acquisition" has been used to describe a variety of approaches to acquiring information for expert systems.[1] Some involve the creation of new knowledge bases; others permit incremental additions or debugging of an existing one. Some acquisition systems are simply syntax-driven tools to help the knowledge engineer enter new information; others autonomously deduce what information to add.

The approach taken by this work is described in the following section. Related approaches to knowledge acquisition, and techniques similar to those used by $K^n{}_{A}c$, are presented in Section §2.

§1. The $K^n{}_{A}c$ Approach

Knowledge acquisition systems vary in both their objectives and in their methodologies for accomplishing these objectives. This section describes the goal of the $K^n{}_{A}c$ system and present a high level view of the approach taken towards satisfying it.

[1]The term has even been used to describe the acquisition of information *from* such knowledge bases (e.g., [BBD86]).

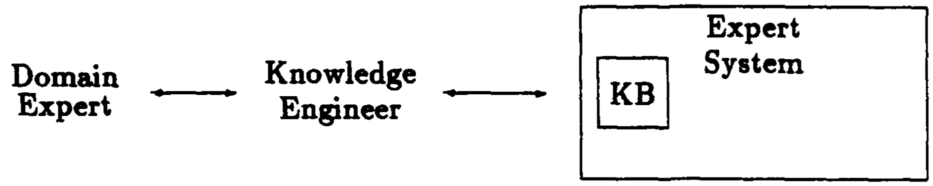


Figure 1: Current Model for Human Experts and Knowledge Engineers.

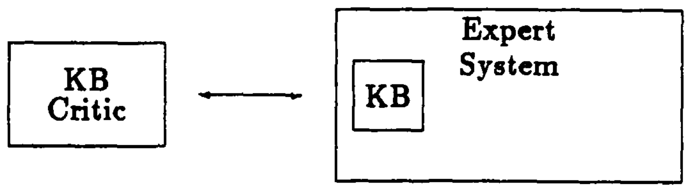


Figure 2: Autonomous Learning through Introspection.

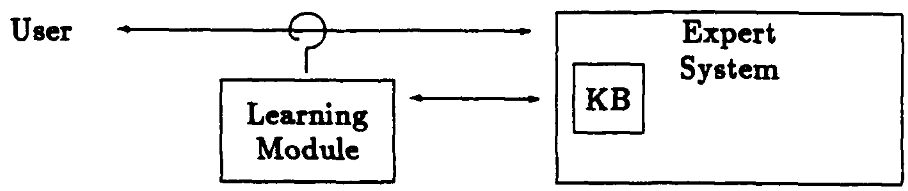


Figure 3: Learning via Normal Usage of the Target Expert System.

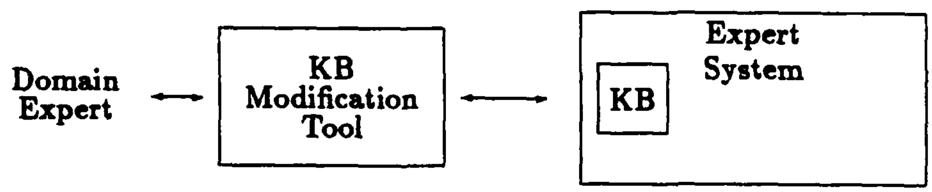


Figure 4: Explicit Modification of the Knowledge Base by the Domain Expert.

§1.1 *The Goal*

To understand the portion of the knowledge acquisition task addressed by the K^nAc system [Lef85], consider the typical means by which knowledge bases are constructed, namely, through dialogs between a domain expert and a knowledge engineer (see Figure 1). In these dialogs, the knowledge engineer attempts to encode information about a particular application domain as provided by an expert in that domain. The knowledge engineer understands the representation scheme used by the knowledge base and has complete access to its contents, but does not have expertise in the application domain. The domain expert, on the other hand, need not be aware of the representation or the extent of the expert system's knowledge base. The goal of this work is the automation of the knowledge engineer's role in modifying the expert system's knowledge base, through dialogs with the domain expert, so as to reflect the expert's view.

Note that the expert's goal is not to modify the knowledge base (as in Figure 4) — this is the knowledge engineer's (or K^nAc's) role. Rather, the expert simply presents information (e.g., tasks, objects, constraints, etc.) about the application domain. This information must be integrated into the existing knowledge by the knowledge engineer.

If the role of the knowledge engineer is viewed as an interface between the domain expert and a knowledge base, this interface may be thought of as a layered system. Neither the layers closest to the domain expert (e.g., a natural language front-end, a discourse manager, a graphical interface) nor those at the knowledge base end (e.g., knowledge base accessor functions) are the primary focus of this project; rather, K^nAc addresses those layers responsible for modifying the knowledge based on the information provided by the expert and for determining

what additional information needs to be obtained. Accepting and presenting this information in a "user friendly" format (natural language, graphics, etc.) is beyond the scope of this project; a natural language parser/generator along with a discourse manager, tuned to knowledge acquisition dialogs and the office automation domain, is being developed concurrently (see [WPML84]).

The separation of the $K^{n}{}_{A}c$ system from a particular style of user interface, be it natural language or graphics or some other type of interface, and from a particular knowledge base representation is intended to 1) maintain a degree of system portability and 2) bound the scope of the project to those issues directly concerned with the acquisition process. The degree to which the acquisition system depends upon the front-end and knowledge representation scheme is discussed in Chapter 4.

§*1.2 The Methodology*

$K^{n}{}_{A}c$'s primary function is to compare an expert's view of a domain against that contained in an existing knowledge base and to modify the knowledge base so that it better reflects the expert's perspective. To accomplish this, $K^{n}{}_{A}c$ must be able to match partial entity descriptions (e.g., tasks, objects, etc.) provided by the expert to those in the knowledge base, recognize the discrepancies between the two and make the implied modifications.

In order that $K^{n}{}_{A}c$ be able to engage in a dialog with the expert, the system must attempt to integrate new information as it is presented. Though some ambiguity may be resolved by delaying the interpretation, the system would not be able to provide timely feedback (e.g., questions, warnings, etc.) if it waited to "post-process" the entire dialog.

The matching problem faced by the system is similar to that found in other recognition systems: "noisy" data (i.e., incomplete or inaccurate) must be integrated into an appropriate (possibly ambiguous) context. Because K^nAc is a knowledge acquisition system, however, the problem is more subtle. Rather than trying to match its input against a "library" of templates (i.e., descriptions of tasks, objects, relations, constraints, etc.), the system is using the information provided by the expert to modify its existing knowledge (i.e., its templates). Perfect matches are not the goal — knowledge acquisition implies modifications to the existing knowledge base. Thus, differences between information coming from the expert and that already known to the system are necessary for these modifications.

Thus, the K^nAc system faces a unique type of matching problem. In comparing information from the expert with that in the knowledge base, the best matches are not necessarily those with the fewest differences, since these differences may represent desired modifications to the knowledge base. If such modifications can be anticipated, however, the best matches would be those containing the fewest *unexpected* differences. Hence, the K^nAc system must provide the capability for such context dependent matching and a means of anticipating modifications. These subsystems are described in detail in Chapters 5 and 6.

§2. Related Approaches

As the need to construct substantial knowledge bases has become apparent, systems to simplify this task have been developed. In general, knowledge acquisition involves using the information obtained from some source to modify some body of knowledge. There is, however, great variation in the sources of informa-

tion, the methods of obtaining it, and the use(s) to which it is put. Consider the following dimensions along which knowledge acquisition systems may vary:

- *What types of information are provided?* Some knowledge acquisition systems provide a convenient way for the user to explicitly modify the knowledge base (e.g., TKAW [KBJD86]). Others may be told about the application domain through examples or training sequences (e.g., SEEK2 [GWP85]) or through normal use of the target expert system (e.g., LEAP [MMS85]) and deduce the appropriate modifications. Some systems correct, or assist the user in correcting, errors in the knowledge base (e.g., Teiresias [DL82]). These errors may either be detected by the user or by the system itself.

- *At what point in the expert system's life cycle does the knowledge acquisition occur?* Knowledge acquisition systems may be used as early as the initial development stage of an expert system (e.g., to select an appropriate representation scheme, algorithm or control strategy) or as late as helping the end-user interact with the finished product. They can aid in the initial building of a knowledge base or in the refinement of an existing one (e.g., SEEK2, MOLE [EEMT86]).

- *How is the knowledge provided?* A knowledge acquisition system may be a passive tool used by knowledge engineer, or it may actively extract information from a domain expert. This may involve an interactive dialog between the user and the system (e.g.,Teiresias, TKAW), filling in knowledge structures (e.g., [MMW85]), providing or evaluating examples of correct and/or incorrect results, or replying to a series of questions from system.

- *How much autonomy does the system have?* At one end of the spectrum are tools that assist the knowledge engineer in modifying and debugging a knowledge base. At the other extreme are systems that autonomously modify a knowledge base using either examples (cases), consistency (or other) constraints, or other "learning" techniques.

- *Where does the system get its "expertise"?* How much does the knowledge acquisition system have to know about the application domain to provide assistance? How much knowledge about the knowledge representation is required? Must it "understand" the expert system's functionality and objectives? Finally, what should it understand about the knowledge acquisition process itself?

The following sections present a brief view of some approaches to knowledge acquisition, their treatment of these issues, and their relevance to this work. In addition to these alternative approaches, several other projects, while not directly addressing the issue of knowledge acquisition, relate closely to $K^{n}{}_{A}c$'s approach. For instance, work on partial matching of complex structures, intelligent interfaces, dialog controllers and computer-based tutoring systems all have some bearing on this project.

§2.1 *Knowledge Base Refinement*

Attempts to make expert systems more robust and capable of "common sense" reasoning have lead to the need for significantly richer knowledge bases. This endeavor has lead to more powerful knowledge representation languages and more sophisticated knowledge structure editors.

Perhaps the most ambitious attempt at constructing a complex knowledge

base is being undertaken by the CYC [LPS86] project. As part of a decade-long attempt to encode the contents of a one-volume encyclopedia, a rich but general knowledge representation based on RLL [GL80] and KRL [BW77] has been developed and a knowledge acquisition methodology specified. This methodology provides a means of entering new structures via a "frame" editor that guarantees that the structure is completely and consistently specified by guiding the expert through the slots of the new structure. It also provides a Copy&Edit facility to permit new knowledge to be defined in terms of existing entities.

This philosophy of using the existing knowledge to assist in the acquisition of new information is very much in line with that of the $K^n{}_Ac$ project. As stated in [LPS86],

> "As the size of the knowledge base grows, it becomes increasingly likely that one can find a match that's close enough to result in a large savings of time and energy and consistency."

While the likelihood of such a match existing increases with the size of the knowledge base, it is unclear that one's ability to *find* this match will also increase. The CYC project recognizes the difficulties posed by such a large knowledge base:

> "The expert makes most of these connections by pointing to related frames, as s/he navigates through 'knowledge space.' This navigation can be implemented simply by a Zoglike network of menus or less simply by using three-dimensional graphics, joysticks, helmets, and even less simply by employing artificial personae as guides."

As an "expert" in the process of knowledge base modification, $K^n{}_Ac$ may be seen as playing the role of "knowledge base guide".

Though $\mathrm{K^{n}{}_{A}c}$ and CYC have similar goals, their approaches are somewhat different. While CYC's "application domain" (e.g., the contents of an encyclopedia and the common sense knowledge required to understand it) causes some blurring of the distinction between "domain experts" and "knowledge engineers" (or "knowledge enterers", to use their terminology), the knowledge acquisition process consists of people actively editing the knowledge base. The CYC system presents tools to aid in this modification. With $\mathrm{K^{n}{}_{A}c}$, on the other hand, the domain expert is more insulated from the knowledge base. It is the system that bears the responsibility for integrating the expert's knowledge.

Another potentially powerful technique mentioned in the CYC work (as well as in [Car83]) is the idea of using *analogies* to determine what entities to modify and how to change them. While analogies, per se, are not used explicitly in $\mathrm{K^{n}{}_{A}c}$, this technique is closely related to idea of "context-dependent matching" discussed in Chapter 5 Section §2.2.

Another approach to updating a system's knowledge base relies on a debugging paradigm. It assumes that the user is attempting to correct a problem in the existing system. Teiresias [DL82] is a knowledge acquisition system developed for updating MYCIN's [Sho76] knowledge base. It enables a human expert to monitor and correct the performance of the underlying expert system. It helps the user identify the cause of an error by providing a means of tracing back through the system's actions that led to the error. Teiresias then allows the expert to add or modify the information necessary to correct the error.

While both Teiresias and $\mathrm{K^{n}{}_{A}c}$ assist in the process of updating an expert system's knowledge base, Teiresias is built around the idea of debugging an error in the existing knowledge; it is only invoked when an error has been identified. While errors (inconsistencies, missing information ("holes"), constraint viola-

tions, etc.) are one means of anticipating modifications to the knowledge base, they are not the only means. The K^nAc approach does not confine the acquisition process to be error-driven. This permits the acquisition system to be used during the initial development of the knowledge base, not just to debug an already functioning system.

Furthermore, Teiresias does not address the question of what modifications should be made to the knowledge base. This task is left almost entirely to the human expert. The error must be identified *by the user*, and *the user* must provide the correction. One of the major goals of this work, on the other hand, is the ability to decide how to modify the knowledge.

Finally, much of the aid that Teiresias provides is guiding the user through the knowledge base in order to identify and correct a known error. Though Teiresias is not confined to the knowledge representation structure used by MYCIN (production rules), much of its power seems possible because of this relatively simple structure. It is not clear that this approach would do well in an environment with more complex structures nor that the power inherent in such structures would be taken advantage of.

The MORE knowledge acquisition system [KNM85] uses a somewhat more sophisticated domain model. It attempts to build and strengthen the set of rules used in the MUD diagnostic system [KM85], by obtaining rules and weights from the user, checking for "weakness" in the diagnostic capabilities of a set of rules, and detecting potential inconsistencies in the numeric values assigned by the user. Instead of simple condition-action pairs (i.e., production rules), MORE's domain model contains *hypotheses*, *symptoms*, *conditions*, *links*, *paths*, *tests* and *attributes*. By having this additional semantics as part of the knowledge representation, it can use more sophisticated strategies such as *differentiation* (to acquire

a distinguishing symptom for a pair of hypotheses) and *symptom distinction* (to refine symptoms) to increase the diagnostic power of the knowledge.

As with Teiresias, MORE uses the state of the existing knowledge base to determine what modifications are necessary. Since the state of the knowledge is one of the means by which $K^n{}_A c$ generates expectations, the strategies used by MORE are of interest to this work. Since $K^n{}_A c$ is not restricted to acquisition for diagnostic systems, however, not all of these strategies are directly applicable. Some of $K^n{}_A c$'s "state of the knowledge" heuristics are similar to the techniques used by MORE.

§2.2 *Learning and Teaching*

Though the boundary between "learning" and "knowledge acquisition" is sometimes blurred, a distinction may be drawn between systems that autonomously modify a body of knowledge and those in which an external source is responsible for such change. The types of learning of particular interest to this research may be described as "learning by being told" and "learning by asking".

Whereas systems such as Lenat's AM [DL82] concentrated on autonomous learning (discovery), we are more concerned here with the interactive process of extracting information from a human expert. While $K^n{}_A c$ is not concerned with autonomous learning per se, the techniques that enable such learning to occur are of definite interest. Examination of an existing knowledge base to detect incompleteness or inconsistencies (or other "areas of interest") provide another powerful means by which $K^n{}_A c$ may anticipate knowledge base modifications.

Between completely autonomous and totally passive learning systems are those in which the learning component is given a set (or sets) of examples (or

"cases") from which it must generate appropriate modifications. LEX [MUB82], for instance, acquires and modifies its set of heuristics by generating test problems for its underlying expert system to solve and then analyzes the solution path taken. This approach requires several complex components: 1) a "problem solving" module which is actually part of the underlying system rather than part of the knowledge acquisition system, 2) a "critic" which analyzes the solution, 3) a "generalizer" which modifies the knowledge base according to the critic's analysis, and 4) a "problem generator" which is responsible for providing test cases that will (gradually) expand the system's knowledge base.

Whereas systems such as SEEK2 [GWP85], another system for refining a set of rules, require a data base of cases, the LEAP system [MMS85] gets its training examples from the normal use of its underlying VLSI design expert system. In a given situation, the expert system provides the user with a set of options based upon its existing set of rules. If the user chooses to override these options, LEAP recognizes this as a potential learning situation. It formulates a new rule to capture the user's action and then generalizes this rule using an "explain-then-generalize" strategy. The "explanation" capability is based on a model of the domain theory.

A similar approach is taken by Odysseus [Wil86] which attempts to debug and refine a knowledge base for the Heracles [Cla85] expert system (a generalized version of NEOMYCIN [Cla84]). This system observes the normal problem solving behavior of the domain expert and attempts to justify the expert's actions based on its existing knowledge base. Note that this "justification" relies only upon the existing rule-base; it does not attempt to generate new rules by relying on a deeper "domain model". Further, Odysseus is not trying to directly match the expert's action to something in its knowledge base; rather, it is trying to

justify the action by constructing a plausible "line of reasoning". If it determines that no such justification can be generated, it attempts to select the portions of the knowledge base responsible for the failure.

Although these systems approach knowledge acquisition from a somewhat different direction than does $K^n_{A}c$, both they and $K^n_{A}c$ compare the expert's knowledge (either as deduced from the expert's actions or as presented to the system) with the existing knowledge base and infer the required modifications from the resulting differences. Because of the different underlying knowledge representations, the manner in which the expert's information is presented, and the assumptions made about how knowledge is to be used, the processes by which these differences are detected and the necessary modifications inferred diverge. These systems tend to work with rule-based knowledge; the differences (between the expert's and the system's view) usually arise from the failure of the system, via chaining of these rules, to produce the desired (i.e., the expert's) results. $K^n_{A}c$, on the other hand, tries to compare the expert's model of the domain with the system's, using the general frame-structure matching techniques described in Chapter 5.

From a very different perspective on learning, the knowledge base may be viewed as a network of highly interconnected concepts. Thus, the recent revival (and apparent success) of low-level, autonomous learning in networks of simple, interconnected processing elements (e.g., [Bar85]) raises interesting possibilities for autonomous learning. Part of the knowledge base modification process could occur on a distributed, localized fashion. This approach, however, is not currently part of the $K^n_{A}c$ project.

Work in intelligent tutoring systems [Woo83] provides an interesting parallel to the problem of knowledge acquisition. In both cases, a dialog between a human

and an intelligent system takes place for the purpose of transferring knowledge. In tutoring systems, the system is the "expert" and the human is the "student"; knowledge acquisition reverses these roles. By examining the teaching strategies used in tutoring systems, some insights into the way in which the domain experts may present information were developed.

Finally, Piaget's studies of the learning process in children should also be considered. An important theme throughout his work is the role of *assimilation* and *accommodation* in the learning process. He defines *assimilation* quite broadly as "the integration of any sort of reality into a structure." [Pia64] This existing structure may "remain unaffected or else be modified to a greater or lesser degree by this very integration." [Pia71] The modification or restructuring of the existing knowledge is what Piaget calls *accommodation*.

Though no claim is made for the psychological validity of the type of "knowledge assimilation" that occurs in K^nA_c, the overall concept is very much in the spirit of the assimilation process described by Piaget. K^nA_c's attempt to match new information to what is already known and to integrate this new information into existing knowledge structures in order to improve the quality and breadth of a knowledge base nicely mirrors Piaget's view of assimilation as described in [BHR75]:

> "Assimilation involves a thought process and a degree of mental analysis. The subject has to recognize that the new situation has similar features to the past experience and he then thinks about the new in terms of the past... Assimilation strengthens knowledge gained by previous experience; it is the *integration* of any sort of reality into a thought structure, and it is this assimilation which seems to me to be fundamental in learning."

§2.3 *Knowledge Base Interfaces*

Intelligent interfaces and knowledge acquisition systems are related in several ways. A knowledge acquisition system may be thought of as an interface between a domain expert and a knowledge base. More traditionally, however, user interfaces may be viewed as a translation between a user's desired action and an underlying system's equivalent functionality. There is generally some type of mapping used to accomplish this. Knowledge acquisition in these systems often consists of modifying these mappings. The importance and difficulty of this task increases as these structures become increasingly complex.

Consul [Wil81] is an interface system designed to simplify a user's interaction with a collection of software tools. It contains descriptions of the user's actions and of the tools' functionality. By providing a mapping between a user's view of an action and the system's (i.e., the collection of tools') actual capabilities, Consul finds the appropriate tool actions needed to carry out a desired user action.

Consul contains a knowledge acquisition facility to allow the user to provide additional descriptions of tools and user actions. When these descriptions are added to the system, they must be interactively classified into Consul's knowledge base. This must be done without assuming expertise on the part of the user with regard to the internal workings of the Consul system or the content of its knowledge base. The system guides the user through a classification dialog by using the structure and content of the knowledge base. It shields the user from the internal representation of these tool/action descriptions and is able to prompt the user by making use of the information already known to the system. $K^{n}{}_{A}c$ attempts to take this one step further and strives to relieve the user of the task

of integrating the domain knowledge presented.

Another interface between a user and a set of tools is the POISE system [CLLH82]. Rather than modeling tool invocations, POISE contains descriptions of what a user might try to accomplish in a particular domain. Its domain model includes descriptions of tasks, objects, relationships, constraints, and goals. $K^{n}{}_{A}c$ was originally developed to assist in expanding POISE's description library.

Adding information to a system with a knowledge base formalism as rich as POISE's is more complex than generating new production rules. However, this same richness enables the acquisition system to take advantage of the semantics embedded in the knowledge representation without having to be aware of the semantics of a particular application domain. With task descriptions, for example, $K^{n}{}_{A}c$ can make use of the relationship between a task and its sub-tasks without having any inherent knowledge about specific tasks for a particular application.

§2.4 *Knowledge Acquisition Dialogs*

Several existing expert systems use a knowledge acquisition dialog to build or modify their knowledge bases [Ben83,DL82]. Most of these systems, however, tend to place the initiative either entirely in the knowledge acquisition system or in the domain expert. In the first case, the dialog tends to be quite rigid and stylized (e.g., obtaining values for fields of fixed data structures). In the latter case, the system will accept and respond to individual user commands and requests, but is completely passive.

The KLAUS system [HH80] also uses a dialog to augment its knowledge base and addresses many of the same issues as $K^{n}{}_{A}c$ does. These include: 1) a "learning by being told" approach to knowledge acquisition, 2) the use of mixed-

initiative dialogs with a domain expert unfamiliar with the the knowledge representation system, 3) the integration of the information provided by the user into the existing knowledge base, and 4) recognizing (and requesting) "missing" information.

KLAUS places a heavy emphasis on the natural language capabilities of the system and on the simultaneous acquisition of linguistic and conceptual information. Its knowledge base is a class (or part-of) hierarchy, represented in a first-order logic. The knowledge acquisition, therefore, is a classification problem; it attempts to correctly locate new concepts in the knowledge "tree", obtaining the necessary distinctions and refinements from the user.

A dialog manager developed for the Aquinas knowledge acquisition workbench [KB86] places greater emphasis on the interaction between the user and the system than on the natural language aspects of the discourse. By examining how experts used the Aquinas system to construct knowledge bases, a set of heuristics was developed. A hierarchy of heuristics for knowledge acquisition is proposed and includes categories such as *temporal reasoning*, *complexity* of the knowledge base, knowledge base problems, user preferences, etc. Although the more detailed levels of the hierarchy appear specifically aimed towards the Aquinas system (e.g., heuristics involving "rating grids"), the more general categories overlap to a large extent with the heuristics used by $K^{n}{}_{A}c$. (See Appendix B for the complete listing of $K^{n}{}_{A}c$'s heuristics. The similarities of heuristics (or at least classes of heuristics) developed for knowledge acquisition systems for two very different types of knowledge bases bodes well for the domain (and knowledge representation) independence of these heuristics. Exploring the extent to which these heuristics must rely upon the particular target system (or application domain) is one of the goals of this work.

Approaching the problem from a somewhat different tack, [BBD86] examines discourses in which users interact with an intermediary in order to *acquire* information from a knowledge base. It attempts to construct a model of the (human) intermediary by analyzing these "knowledge elicitation" discourses. Recognizing that not all the required information is provided explicitly by the user during the discourse, other sources of knowledge (e.g., a user model, structure of the discourse, etc.) are explored.

While $K^n{}_Ac$ is not concerned with natural language discourses *per se*, it is clear that the structure of these discourses between the domain expert and the intermediary, be they in natural language or some other format, is a potentially rich source of information about the knowledge acquisition process. The extent to which this information may be used by $K^n{}_Ac$ depends primarily on the sophistication of the discourse manager (see Chapter 4 Section §6.) being used. Because $K^n{}_Ac$ is not currently integrated with a very sophisticated front-end, minimal discourse information is available to the system. (See Appendix §3..)

Chapter 3

Interviewing the Domain Expert

K^nAc's approach to knowledge acquisition is modeled after interviews between a domain expert and a knowledge engineer. As part of the ongoing development of POISE's knowledge base, a series of interviews between a domain expert in the office environment (i.e., the department's principal clerk) and a knowledge engineer familiar with the POISE system were conducted. These interviews were examined to gain insight into the types of knowledge base modifications being made and to explore the knowledge engineer's role.

This chapter describes the set of interviews examined, shows how they were formally modeled, traces a portion of one such interview, and presents the implications of these interviews for automated knowledge acquisition.

§1. The Interviews

In order to understand the various techniques used to acquire information under different circumstances, several types of knowledge acquisition interviews were conducted. They varied in the scope and complexity of the information being sought, the extent of the interviewer's prior knowledge of the task to be learned, and the degree of initiative taken by the interviewer. By obtaining

a collection of different approaches to interviewing the domain expert and an understanding of when to use each of these techniques, an intelligent knowledge acquisition system can better tailor its strategy to individual situations.

Three interviews have been examined to date. The first concerns the funding of graduate students; the second, presented in Section §4., concerns reimbursement for travel expenses; the third involves the hiring of a new faculty member.

One dimension along which the three interviews varied was the complexity of information being sought from the domain expert. The first interview was intentionally broad in scope. It attempted to codify an entire process rather than a particular individual's (here, the principal clerk's) task. In the latter interviews, the emphasis was placed on the clerk's role in the execution of the procedure.

In order to examine a dialog involving less clearly defined objectives, as is common early in the building of an expert system's knowledge base, the interviewer did not know the topic of the third interview beforehand. This interview displayed characteristics of pattern-matching recognition as the interviewer tried to find an appropriate context in which to assimilate the information provided by the clerk, and then appeared to be goal-directed in an attempt to confirm (or deny) hypothesized contexts.

In this series of interviews, the extent to which the interviewer maintained the initiative was also observed. The first interview was fairly broad in scope and the domain expert tended to control the dialog with minimal guidance supplied (or required) by the interviewer. In the second and third interviews, because a specific role was to be defined, the interviewer needed to exert more control in order to acquire this information.

§2. Modeling the Interview

To guide the initial development of the K^nA_c system, a methodology for examining these interviews was needed. By viewing the interviews as a means of transforming the knowledge base from one "state" to another (presumably more complete or correct) one, a series of such states could be modeled in terms of POISE's knowledge representation formalism. The modifications occurring between successive states could then be enumerated and the possible causes for these modifications explored. This approach provided the insights into the knowledge engineer's role in these acquisition dialogs that formed the basis of the K^nA_c system.

Since it was not possible to monitor the knowledge engineer's mental state during the course of a knowledge acquisition dialog, the knowledge engineer was debriefed before and after several of the interviews. This provided an initial domain model (i.e., the knowledge engineer's view of a portion of the domain prior to a particular interview); the knowledge engineer's modified version of the knowledge base served as a final model. The interview was then segmented to provide intermediate knowledge base states.

The issue of how to segment the interview in a reasonable fashion arose. Since the objective was to identify and understand the modifications made to the domain model, the granularity with which the "interim states" were selected depended upon the size of the changes to be examined. Segmenting the interview at each ply of the discourse (that is, at each change of speaker) was often too large a granularity; a single "question" or "answer" may contain too many relevant modifications. (Occasionally, the reverse is true: a speaker may convey little useful information during a ply.) Manually segmenting the dialog (i.e., rec-

ognizing "chunks" that result in changes to the knowledge base) even without an exact understanding of the desired types of changes, was not difficult. The automation of this segmentation process is examined in Chapter 6 Section §3.

§3. The Representation Formalism

The objective of the knowledge acquisition process is the formulation of a model of the expert's task in the formalism of the target expert system. The first step in our approach to understanding this process was to use this formalism not only for the final results of the interview, but for initial and interim models of the task as well. This presumes that the expert system has some initial model of the task (and its associated goals, subtasks, tools and objects) which may be refined as the interview progresses. The initial and interim models are often incomplete, inconsistent or incorrect. If these are to be described in the same language as the final model, the target formalism must permit such "imperfect" descriptions.

The POISE system provides a rich language [CL82] for describing procedural tasks, their goals, and the objects they utilize. It incorporated both a procedural and an object-centered view of these tasks. In the domain of office automation, for example, tasks such as communicating information (e.g., via electronic mail) and filling out forms are described, as well as the relevant objects involved, such as messages, forms, and mail systems.

The procedural descriptions are hierarchical, with each description's IS clause containing the temporal ordering of its constituent procedures (using an extended shuffle expression grammar [BW82]). Attributes and objects related to the procedure are defined in the WITH clause and constrained in the COND clause. The

```
PROC     PURCHASE-ITEMS
DESC     "Procedure for purchasing items with non-state funds."
IS       (RECEIVE-PURCHASE-REQUEST
         ' (PROCESS-PURCHASE-ORDER |
            PROCESS-PURCHASE-REQUISITION)
         ' COMPLETE-PURCHASE)
WITH     ( (Purchaser = RECEIVE-PURCHASE-REQUEST.Form.Purchaser)
           (Items     = RECEIVE-PURCHASE-REQUEST.Form.Items)
           (Vendor    = RECEIVE-PURCHASE-REQUEST.Form.Vendor-name))
COND     (for_values Purchaser Items Vendor-name
            (eq RECEIVE-PURCHASE-REQUEST.Form
                PROCESS-PURCHASE-ORDER.Form
                PROCESS-PURCHASE-REQUISITION.Form
                COMPLETE-PURCHASE.Form))
PRECONDITIONS —
SATISFACTION    (for_values Purchaser Items Vendor
                    (exist COMPLETE-PURCHASE.Form))
```

Figure 5: A POISE Procedure Specification

PRECONDITION and SATISFACTION clauses defined the state of the "world" (i.e., the semantic database) before and after the procedure occurred. Figure 5 shows an example of a POISE procedure taken from the purchasing domain.

POISE represents objects in a frame-like semantic database based on SRL [WF83]. The descriptions include attributes of the objects, pointers to the procedures in which they are used, and specialization/generalization links to other objects. Although the formal representation of objects was never completely specified in the POISE project, a more uniform representation for its event and object descriptions was explored in [BC85].

The POISE representation formalism was slightly modified for use with the

K^nA_c system. Most of the modifications were made in order to organize the information in a more uniform and a more perspicuous manner. These modifications include:

- *Representing events, objects and relationships uniformly.* In K^nA_c, events, objects and relationships are all specializations of the more generic knowledge structure *knac-structure*. This permits most portions of the acquisition process to handle all of these structures in a uniform manner and only provide special techniques for the unique aspects of each.

- *Distinguishing between* is-a *and* part-of *hierarchies.* In POISE, the IS clause of *event* structures contained the steps (i.e., sub-events) that comprised each event. Specializations of an event were represented as an event that contained the more general event as its only step. For instance, RECEIVE-PURCHASE-REQUEST contained RECEIVE-INFORMATION in its IS clause and had a COND clause that constrained the information received to be a purchase-request. The existence of two distinct fields in K^nA_c, *generalizations* and *parts*, eliminates the confusion between the components of an event (or object or relationship) and its generalizations.

- *Treating relationships as full-fledged entities.* In POISE, relationships were considered to be fields of an event (or object). They were not entities in their own right and the descriptions was somewhat ad hoc. K^nA_c promotes relationships to an equal footing with events and objects, permitting richer (and more uniform) relationship descriptions.

- *Grouping all constraints within each description.* Various fields of each structure description may be viewed as collections of constraints. In an event, for example, the *temporal-relationships* are constraints on the order-

ing of the steps of that event; the definitions contained in the *attributes* field may be viewed as equality constraints; the *pre-* and *post-conditions*[1] are constraints on the state of the knowledge base at a particular time; the *constraints* field is a catchall for any other constraints. While $K^{n}{}_{A}c$ permits these constraints to be specified separately for the sake of perspicuity, it (internally) gathers them together in order to maintain consistency among the various types of constraints. (Chapter 5 Section §1.2 discusses $K^{n}{}_{A}c$'s handling of constraints.)

- *Providing a "version" mechanism.* Although POISE permitted event or object descriptions to be *instantiated* in multiple contexts, it made no provision for multiple versions of the descriptions themselves. The modification of these descriptions during the knowledge acquisition process requires a mechanism for maintaining multiple versions of each description.

Thus, all descriptions in $K^{n}{}_{A}c$'s knowledge base are specializations of the generic KNAC-STRUCTURE. As shown in Figure 6, each structure contains fields[2] that: identify the structure (name, icon, synonyms, description); locate the structure in an IS-A hierarchy (generalizations, specializations); locate the structure in a PART-OF hierarchy (parts, part-of); define features of the structure (attributes); constrain various aspects of the structure (constraints). Information about each slot, or *facets*, constrain the values permitted in each field. These facets restrict the type and number of values permitted and declare whether such values are required.

In addition to the fields common to all *knac-structures*, *event* descriptions, shown in Figure 7, contain *temporal relationships* and *causal relationships* among

[1] $K^{n}{}_{A}c$'s *post-conditions* correspond to POISE's *satisfaction* clause.

[2] Italicized fields are for $K^{n}{}_{A}c$'s internal use only.

Field	Facets
name	(multiple-value? . nil) (required? . t)
icon	(multiple-value? . nil)
synonyms	(multiple-value? . t)
description	(multiple-value? . nil) (range . string)
creation-time	(multiple-value? . nil) (required? . t) (range . number)
last-modification-time	(multiple-value? . nil) (required? . t) (range . number)
generalizations	—
specializations	—
parts	—
part-of	—
attributes	—
attribute-names	—
constraints	—

Figure 6: KNAC-STRUCTURE Description

their sub-events (i.e., their *parts*). While it would be possible to include these under the more general heading of *constraints*, they often play such a crucial role in the processing of *events* that this distinction, originally made by POISE, was maintained. *Events* also contain pointers to *associated objects*, that is, those *object* entities manipulated by or manipulating this *event*. The *pre-* and *post-conditions* contain constraints on the state of the knowledge base necessary for the event to begin and upon its completion, respectively. Additionally, the *generalizations*, *specializations*, *parts* and *part-of* fields are constrained to contain *events*.

In addition to the generic *knac-structure* information, an *object* description, shown in Figure 8, contains *spatial relationships* among its parts and pointers to *associated events*. The *generalizations*, *specializations*, *parts* and *part-of* fields are constrained to contain *objects*.

A *relationship* description, shown in Figure 9, contains the name of the *inverse* and *converse* relationships, where applicable, restrictions on the *domain* and the *range* of the relationship, an associated *demon*, and a description of the *algebraic properties* of the relationship. These properties are used in the comparison of constraints and are described more fully in Chapter 5 Section §1.2.

§4. A Knowledge Acquisition Interview

This section examines a knowledge acquisition interview extracted from a series of discussions between a POISE knowledge engineer and the Computer and Information Science department's principal clerk. (See Appendix A for a complete transcript of this interview.) Prior to this interview, POISE's knowledge

Field	Facets
generalizations	(range . event) (multiple-value? . t) (cardinality . *event-superclass-count*)
specializations	(range . event) (multiple-value? . t) (cardinality . *event-subclass-count*)
parts	(range . event) (multiple-value? . t) (cardinality . *event-step-count*)
part-of	(range . event) (multiple-value? . t) (cardinality . *event-step-of-count*)
attributes	(multiple-value? . t) (cardinality . *event-attribute-name-count*)
constraints	(multiple-value? . t)
temporal-relationships	(range . temporal-relationship) (multiple-value? . t)
causal-relationships	(range . causal-relationship) (multiple-value? . t)
associated-objects	(range . object) (multiple-value? . t) (cardinality . *event-object-count*)
precondition	(multiple-value? . nil)
postcondition	(multiple-value? . nil)

Figure 7: EVENT Description

Field	Facets
generalizations	(range . object) (multiple-value? . t) (cardinality . *object-generalization-count*)
specializations	(range . object) (multiple-value? . t) (cardinality . *object-instance-count*)
parts	(range . object) (multiple-value? . t) (cardinality . *object-part-count*)
part-of	(range . object) (multiple-value? . t) (cardinality . *object-part-of-count*)
attributes	(cardinality . *object-descriptor-name-count*) (multiple-value? . t)
spatial-relationships	(range . spatial-relationship) (multiple-value? . t)
associated-events	(range . event) (multiple-value? . t) (cardinality . *object-event-count*)

Figure 8: OBJECT Description

Field	Facets
generalizations	(range . relationship) (multiple-value? . t) (cardinality . *relationship-general-relationship-count*)
specializations	(range . relationship) (multiple-value? . t) (cardinality . *relationship-specific-relationship-count*)
part-*of*	(range . relationship) (multiple-value? . t) (cardinality . *relationship-containing-relationship-count*)
parts	(range . relationship) (multiple-value? . t) (cardinality . *relationship-contained-relationship-count*)
attributes	(default-value . (domain range)) (cardinality . *relationship-descriptor-name-count*) (multiple-value? . t)
algebraic-properties	(multiple-value? . t) (cardinality . *relationship-property-count*)
inverse	(range . relationship) (multiple-value? . nil)
converse	(range . relationship) (multiple-value? . nil)
domain-facets	(multiple-value? . nil) (range . object)
range-facets	(multiple-value? . nil) (range . object)
demon	(multiple-value? . nil)

Figure 9: RELATIONSHIP Description

base contained information about generic office tasks such as filling out forms, and sending and receiving mail. It also contained descriptions of tasks and objects used in purchasing goods (such as desks and computers), as well as some basic knowledge about "traveling" (such as descriptions for airlines, hotels and expenses). The purpose of this interview was to add information about being reimbursed for business related travel.

To examine the interview and the resulting changes to the knowledge base, the dialog is divided into time frames and the content of each frame is represented in terms of the knowledge base formalism. Recall that domain expert is describing a portion of a particular domain (e.g., how to get reimbursed for travel) rather than actively trying to modify a knowledge base. Thus, the type of information presented is of the same form as that already in the knowledge base. The translation from the natural language dialog to this representation formalism was carried out manually.

The "travel reimbursement" interview began as follows:

> CLERK: *"O.K. — on travel. The proper way of doing it, if it's out of state, is that a travel authorization should be issued before the trip."*

This dialog fragment translates, approximately, into the structures shown in Figure 10. As a result of this information, the knowledge engineer added a TRAVEL-AUTHORIZATION object and an ISSUE-TRAVEL-AUTHORIZATION event to the knowledge base. The object was recognized as a specialization of FORM and the event as a specialization of the event AUTHORIZE. In addition, the initial description of the event TAKE-A-TRIP-AND-GET-PAID, shown in Figure 11, was modified to include the new event as its first step and to contain the additional

EVENT EVENT-1
*STEPS: (*ISSUE-TRAVEL-AUTHORIZATION TAKE-A-TRIP*)*
TEMPORAL-RELATIONSHIPS:
*((*ISSUE-TRAVEL-AUTHORIZATION *before* TAKE-A-TRIP*))*
*CONSTRAINTS: ((*DESTINATION *outside-of* STATE*))*
*ATTRIBUTES: ((*TRAVELER ···*) (*DESTINATION ···*))*

EVENT ISSUE-TRAVEL-AUTHORIZATION

EVENT TAKE-A-TRIP

OBJECT TRAVEL-AUTHORIZATION

Figure 10: Discourse Manager Output (Frame 1)

EVENT TAKE-A-TRIP-AND-GET-PAID
*STEPS: (*TAKE-A-TRIP GET-REIMBURSED*)*
TEMPORAL-RELATIONSHIPS:
*((*TAKE-A-TRIP *before* GET-REIMBURSED*))*
CONSTRAINTS: (···)
*ATTRIBUTES: ((*TRAVELER ···*) (*COST ···*) (*DESTINATION ···*))*

Figure 11: Knowledge Base Event (Frame 1)

constraint. The modified event description appears in Figure 12.

The clerk continues:

> "It can be set up afterwards, but accounting likes to have it in before the trip is taken."

The interviewer was told that issuing a travel authorization consists of, at least in part, sending the authorization to the accounting department. A new procedure was created to describe this and a step was added to the ISSUE-TRAVEL-AUTHORIZATION procedure. The new entities are shown in Figure 13 and the modified knowledge base entities appear in Figure 14.

The clerk then explains:

EVENT TAKE-A-TRIP-AND-GET-PAID
STEPS: (<u>ISSUE-TRAVEL-AUTHORIZATION</u> TAKE-A-TRIP
GET-REIMBURSED)
TEMPORAL-RELATIONSHIPS:
((TAKE-A-TRIP *before* GET-REIMBURSED)
<u>(ISSUE-TRAVEL-AUTHORIZATION *before* TAKE-A-TRIP)</u>)
CONSTRAINTS:
(<u>(DESTINATION *outside-of* STATE)</u>
(···))
ATTRIBUTES: ((TRAVELER ···) (COST ···) (DESTINATION ···))

Figure 12: Modified Knowledge Base (Frame 1)

EVENT SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING

OBJECT TRAVEL-AUTHORIZATION

OBJECT ACCOUNTING

Figure 13: Discourse Manager Output (Frame 2)

"And what you do is list: 1) your destination; 2) the date you plan on leaving; 3) the date you are returning; 4) how you plan on going – whether it's plane, bus, private car or whatever; 5) your estimated expenses, and how much of a reimbursement you're getting – whether it's a set amount or whether it's full; 6) the purpose of the trip, and, of course, the account that the money is going to come out of. Then the traveler has to sign that, and, if it comes out of a grant, the P.I. (must sign it); if it's state funds then the department head signs it, but we never get state travel funds. So it had better come out of a grant or a trust fund."

EVENT ISSUE-TRAVEL-AUTHORIZATION
STEPS: (<u>SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING</u>)
ATTRIBUTES: ((TRAVEL-AUTHORIZATION ···))

Figure 14: Modified Knowledge Base (Frame 2)

Here, the interviewer was told the details about filling out a travel authorization form. This resulted in the modification of the description of the TRAVEL-AUTHORIZATION form and the creation of a procedure for filling one out. This procedure was added (to the beginning) of the ISSUE-TRAVEL-AUTHORIZATION procedure and establishes the appropriate attribute relationships. A new object was mentioned, a TRUST-FUND, which is a possible source of money. New conditions were also revealed (e.g., the traveler's name must be sent to accounting). This constraint arose from the consistency requirements in the modified ISSUE-TRAVEL-AUTHORIZATION procedure.

This is translated as shown in Figure 15; the knowledge base modifications appear in Figure 16.

§5. Analysis of the Acquisition Process

By representing the knowledge acquisition dialog as a series of knowledge base states, the resulting modifications could be examined and the underlying reasons for these modifications could be explored. The knowledge engineer may be influenced, for instance, by the source of the information being acquired, the type of information, its form, the reasons for acquiring it, and what is already known. An understanding of what caused these modifications to the knowledge base forms the basis of $K^n{}_Ac$'s expertise in knowledge acquisition. This section examines what knowledge is available to guide such an acquisition system, how this knowledge may be used and what may be accomplished with it.

Four types of modifications to the knowledge base appeared in the protocols examined: *creation* of new concepts, *deletion* of existing ones, and the modifi-

cation of existing entities by the *addition* or *removal* of portions of them. It is common, especially early in the development of a knowledge base, or when extending its scope, for gaps to exist in the interviewer's understanding of the domain. A gap may consist of a missing event, object, etc. and is remedied by the *creation* of an appropriate entity. For example, in the first segment of the interview in Section §4., the concept of a TRAVEL-AUTHORIZATION had not been included in the knowledge base; a TRAVEL-AUTHORIZATION object and an ISSUE-TRAVEL-AUTHORIZATION event had to be created.

Often, parts of the interviewer's domain model proved to be incomplete or inaccurate. Incorrect information had to be removed; incomplete descriptions had to be supplemented. When the ISSUE-TRAVEL-AUTHORIZATION event was recognized as the first step of the event TAKE-A-TRIP-AND-GET-PAID, that latter description had to be modified to include the former event as a step, and constraints had to be added to reflect its temporal position in the task.

In trying to explain the knowledge engineer's modifications, it became clear that the modifications were generated for widely varying reasons. Some modifications were based on the state of a portion of the knowledge base. For instance, where it could be determined that steps of an event were missing, or that constraints were inconsistent, the expected addition or modification would ensue. In the third frame, for example, it was recognized that the SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING event had an (potentially) unsatisfied precondition (i.e., the existence of a TRAVEL-AUTHORIZATION form). This gap is filled by the addition of the event FILL-OUT-TRAVEL-AUTHORIZATION (which guarantees the existence of the form as a postcondition) as a first step.

Other changes seemed to depend more on the modifications that preceded them. If a step was added to an event, a temporal constraint might follow (e.g.,

the addition of ISSUE-TRAVEL-AUTHORIZATION to TAKE-A-TRIP-AND-GET-PAID); when an entity was created, it was often incorporated into an already existing entity (as SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING was added to ISSUE-TRAVEL-AUTHORIZATION) or further described (such as the listing of the parts of a TRAVEL-AUTHORIZATION object).

As described above, knowledge bases get modified for various reasons. For the initial development, or a change of application domain, large amounts of new information were usually required. Debugging errors in the knowledge often resulted in modifications to existing knowledge. Customization of the knowledge for individual users or roles added specialized instantiations of task and object descriptions. By determining the reasons a particular knowledge acquisition dialog (or a part of one) is occurring, the ability to anticipate modifications can be refined.

The information already contained in a knowledge base greatly influenced the acquisition of additional information. By providing a context within which to incorporate the information obtained from the expert, the interpretation of incomplete, potentially ambiguous data was better constrained. Furthermore, incompleteness or inconsistency in the knowledge indicates potential areas on which to focus the acquisition process.

Another consideration in determining how to modify the knowledge was the source of the information. Because different sources may vary in their reasons for providing the information, the types of information they provide, and the resulting modifications, the handling of such information might be different if it comes from a knowledge engineer constructing the knowledge base, an expert in the application domain, an end-user of the expert system, or another expert system.

Last, but certainly not least, the discourse itself provided some insight into what was to be modified. During the discourse, reference was often made to entities already known to the system; focusing on these referenced entities (such as the TAKE-A-TRIP event in the first time frame) provided a context in which to better understand the new information. Matching entities referred to in the discourse to those already in the knowledge base is often non-trivial (such as recognizing the unspecified event in the first time frame, EVENT-1, as TAKE-A-TRIP-AND-GET-PAID); Chapter 5 discusses this matching problem. Other "discourse cues" [WPML84] included explicit topic information (*"O.K. — on travel."*) and context shifts.

These various sources of knowledge acquisition wisdom are all exploited to a greater or lesser degree by the $K^n{}_Ac$ system. The resulting knowledge acquisition heuristics are presented in Chapter 6.

EVENT FILL-OUT-TRAVEL-AUTHORIZATION
ASSOCIATED-OBJECTS: (TRAVEL-AUTHORIZATION)

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *destination*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *departure-date*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *return-date*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *estimated-expenses*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *reimbursement*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *purpose-of-trip*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *source-of-funds*))

EVENT FILL-IN-FORM-FIELD
CONSTRAINTS: ((FIELD = *travel-signature*))

OBJECT TRUST-FUND

OBJECT GRANT

OBJECT P.I.

OBJECT DEPARTMENT-HEAD

RELATIONSHIP IMPLIES
DOMAIN: (SOURCE-OF-FUNDS = GRANT)
RANGE: (APPROVAL-SIGNATURE = P.I.)

Figure 15: Discourse Manager Output (Frame 3)

OBJECT TRAVEL-AUTHORIZATION
PARTS: (DESTINATION DEPARTURE-DATE RETURN-DATE
ESTIMATED-EXPENSES REIMBURSEMENTS
PURPOSE-OF-TRIP SOURCE-OF-FUNDS
TRAVEL-SIGNATURE)

EVENT ISSUE-TRAVEL-AUTHORIZATION
STEPS: (FILL-OUT-TRAVEL-AUTHORIZATION
SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING)
TEMPORAL-RELATIONSHIPS:
((FILL-OUT-TRAVEL-AUTHORIZATION *before*
SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING))

Figure 16: Modified Knowledge Base (Frame 3)

Chapter 4

The K^NAC System

This chapter describes the functionality and architecture of the K^nAc system. The knowledge acquisition protocol presented in Chapter 3 is used to illustrate the system's operation.

At a high level, K^nAc operates by comparing descriptions provided by the domain expert (and translated by the discourse manager) with entities selected from an existing knowledge base. Where the expert's descriptions can be determined to match existing entity descriptions, any discrepancies between them may imply modifications that need to be made to the existing description. If desired, the expert is consulted to verify any such changes. Descriptions sufficiently different from the existing entities are assumed to be new and are added to the knowledge base.

Matching the entity descriptions provided by the expert against those in the knowledge base requires syntactically comparing the structures to determine similarities and differences, evaluating the likelihood of them possibly matching, and determining to what extent the modifications required to make them match are expected. Determining the "expectedness" of these implied modifications relies on the ability of the system to anticipate such changes. These expectations result from a set of heuristics about the knowledge acquisition process.

Figure 17 presents an overview of the $K^n{}_A c$ system. During each cycle of the $K^n{}_A c$ system, descriptions of domain entities are accepted from the user (1)[1] and compared with entities in the existing knowledge base (2). (Figure 18 contains a portion of this knowledge base.) These candidate entities (3) are selected based on $K^n{}_A c$'s expectations of changes to the knowledge base. The comparisons (4) are evaluated both in terms of how well they match and the extent to which the differences between them were expected (5) within the context of the match. Once the best matches are selected, the implied modifications (6) are made to the existing entity knowledge base (7), after being verified with the user (8), if necessary. Expectations of further modifications are generated from a variety of sources, including the information obtained from the discourse (9), the state of entities in the knowledge base (10), previously made modifications (11) and the state of the acquisition process.

The following sections describe the major components of $K^n{}_A c$ the system. To illustrate the system's functionality, the recognition of the task described by the user (EVENT-1) as a modified version of the existing TAKE-A-TRIP-AND-GET-PAID task and modification of the existing description is presented. Specifically, Section §1. describes the selection of candidate matches from the existing knowledge base; Section §2. presents the process used to compare the user's descriptions with the existing entities and how this matching process is tailored to support knowledge acquisition; Section §3. describes the two phases required to evaluate the results of this comparison; Section §4. shows how the knowledge base gets modified as a result of the matches; finally, Section §5. explains how $K^n{}_A c$ generates and manages expectations.

[1]The parenthesised numbers in this paragraph (e.g., (1)) refer to Figure 17.

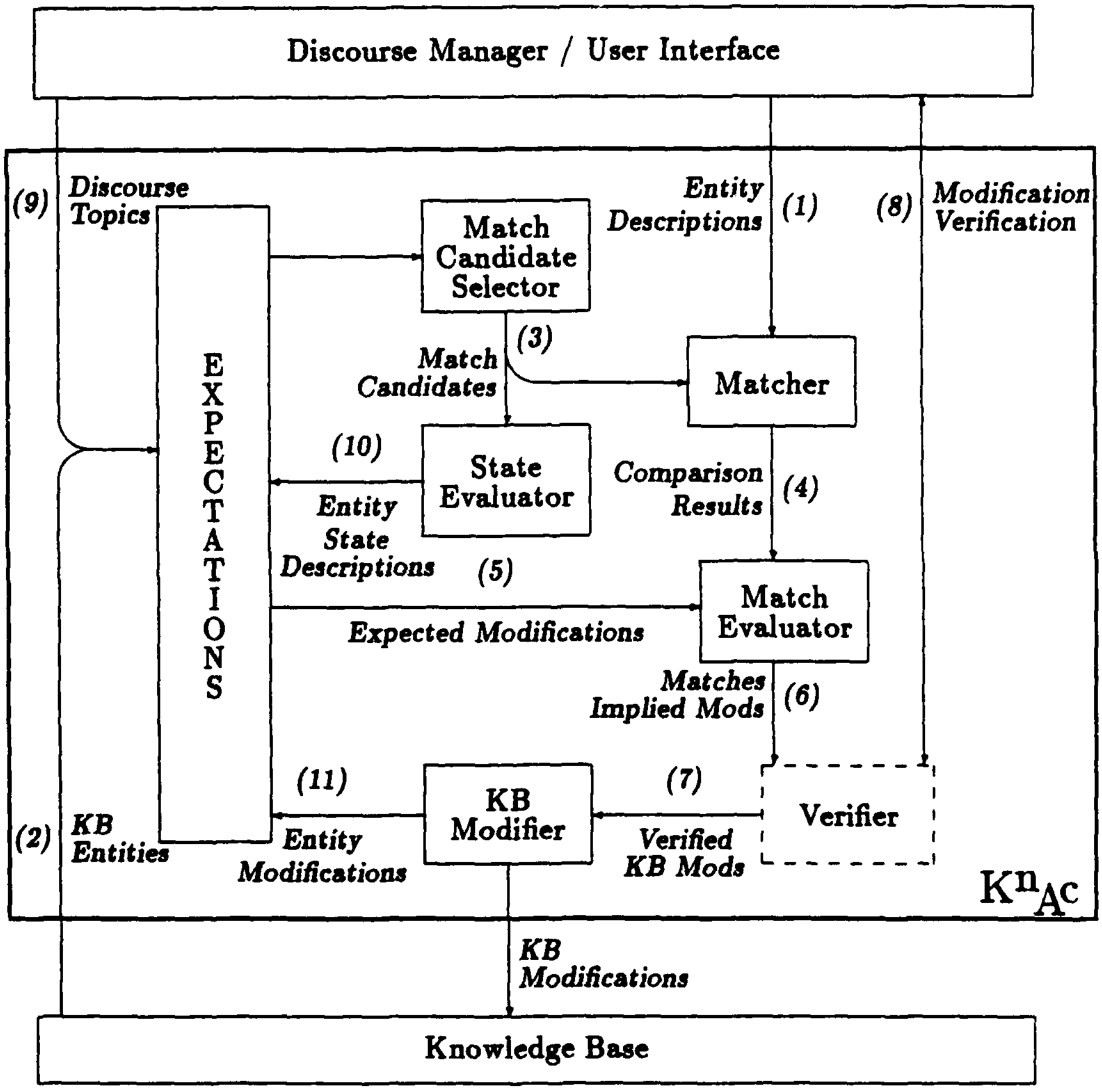


Figure 17: The $K^n{}_{A}c$ System Architecture

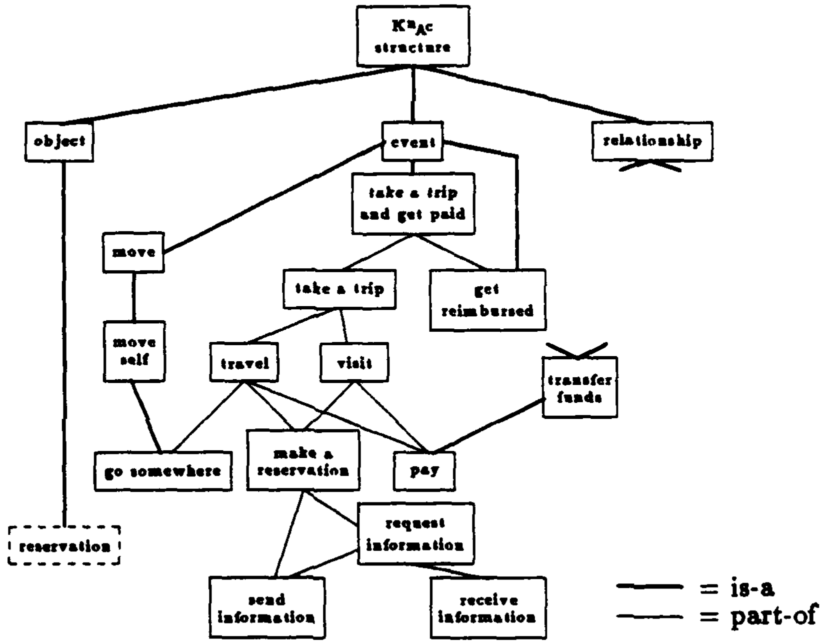


Figure 18: A Portion of the Knowledge Base

§1. Selecting Match Candidates

As will become clear in Section §5. and Chapter 6, a set of expectations of knowledge base modifications is generated by the $K^n{}_Ac$ system during its dialog with the domain expert. The primary use for these expectations is during the evaluation of match results. They are also used to select candidates from the knowledge base against which to compare the discourse entities.

Each expectation contains a "target" entity which will be affected by the modification. Since the set of these "targets" is all the entities that the system expects to be referred to by the expert, these targets are selected as likely candidate match entities. The certainty of the expectation is used as a rating of the candidate.

Thresholding on the certainty of the expectations may be used to reduce the

number of candidates. Since it is possible for an entity to be the target of more than one expectation, the maximum certainty value is used. (See Chapter 6 for a discussion of the rating of related expectations.)

For the initial frame of the discourse, since no interaction with the expert has yet occurred, there are no previous modifications to the knowledge base from which to generate expectations. Thus, only expectations based on the discourse and on the state of the candidate entities are possible. Once the topic of the discourse has been introduced ("O.K. — on travel."), for example, heuristic H_D1[2] (*"Entities close to specified topics are likely to be referenced or modified."*) produces expectations such as:[3]

```
Exp21:  Expecting (certainty 0.100):
  MOD:  ?ACTION ?Value to/from the
                ?Field field of Take_a_trip_and_get_paid
        Derived from Travel and H_D1.
```

Match candidates are derived from this set of expectations. Here, the system produces the following:

Candidate	Rating	Candidate	Rating
TRAVEL	0.3	DRIVE	0.2
FLY	0.2	MAKE-A-RESERVATION	0.2
PAY	0.2	GO-SOMEWHERE	0.2
TAKE-A-TRIP	0.2	TRAVELER	0.2
DESTINATION	0.2	SOURCE	0.2
SEND-INFORMATION	0.1	REQUEST-INFORMATION	0.1
TAKE-A-TRIP-AND-GET-PAID	0.1	KNAC-STRUCTURE	0.1
EVENT	0.2		

Each of these candidate entities are examined for completeness and consis-

[2]See Appendix B for a complete listing of the heuristics.

[3]Values beginning with a question mark (e.g., ?Field) are variables.

tency. Any entities found to be incomplete or incorrect (i.e., inconsistent) will result in expectations being generated. For example, reference to a non-existing entity COST in the event TAKE-A-TRIP leads H_S6 (*"Referred to entities should exist."*) to produce:

```
Exp69:  Expecting (certainty 0.800):
  MOD:  CREATE the Knac-structure Cost
        Derived from Take_a_trip and H_S6.
```

The lack of *parts* (i.e., steps) in the event FLY causes H_S2 (*"Fields with too few components will be augmented."*) to produce:

```
Exp93:  Expecting (certainty 0.480):
  MOD:  ADD ?New-part<is-a-knac-structure-p> to the
            Parts field of Fly
        Derived from Fly and H_S2.
```

An unsatisfied precondition in the step GET-REIMBURSED of the event TAKE-A-TRIP causes H_S3 (*"Unsatisfied preconditions will be satisfied."*) to produce:

```
Exp83:  Expecting (certainty 0.480):
  MOD:  ADD ?New-step<is-an-event-p> to the
            Parts field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S3.

Exp84:  Expecting (certainty 0.480):
  MOD:  ADD (Before ?new-step<is-an-event-p> get_reimbursed)
            to the Constraints field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S3.
```

§2. Comparing Entity Descriptions

In order to determine how to incorporate the information provided by the domain expert into an existing knowledge base, the entity descriptions provided by the expert must be compared to those descriptions already contained in the knowledge base. This requires a means of comparing two entity descriptions. This comparison process differs in several ways from a typical pattern matcher. Since the "goodness" of the match depends not only on how well the two entities match, but also on how they differ (as the following section explains), the comparison process must return this "difference" information as well. Thus, instead of a match simply succeeding or failing, the following match results are possible:

Equal: The user's description exactly matches the candidate knowledge base entity;

Related: The user's description matches an existing knowledge base entity that is related (i.e., linked) to the candidate entity;

Different: After being (recursively) compared to a specified depth, the user's description does not match the candidate entity;

Comparable: The user's description partially matches the candidate entity.

The entity descriptions are structures of various types (e.g., *events*, *objects*, etc.) consisting of fields each of which may contain zero or more values. (Chapter 3 Section §2. contains a description of the representation language.) If two entities are of the same type, they are compared on a field-by-field basis.[4] Since

[4]Even entities of different types are compared using only the fields they have in common. This permits recognising the similarities between an *object* and an *event*, for instance, such as between TRAVEL-AUTHORIZATION and FILL-OUT-TRAVEL-AUTHORIZATION.

the values in a field may be pointers to other entity descriptions, the matching process is recursive. The depth to which this comparison process is taken before deciding the entities are different is specified as a system parameter.

The comparison process for each field depends on the syntax and semantics of that field. For the knowledge representation used in this knowledge base, and for most other frame-like representations, each field may be viewed as either a set of elements or as a collection of constraints. For example, the *parts* field consists of a set of pointers to other entity descriptions; the *temporal relationships* field (of an *event*) contains a collection of temporal constraints on the *parts* of that *event*.

Determining how two sets of elements match and differ is fairly straightforward. In addition, the likelihood of one set being modified so as to match the other may also be determined by considering the size of the sets and range of their elements. Comparing constraints is somewhat more difficult. Parts of this process may be mechanized, however. First, sets of constraints involving the same relationship (such as (TAKE-A-TRIP *before* GET-REIMBURSED) and (ISSUE-TRAVEL-AUTHORIZATION *before* TAKE-A-TRIP)) may be compared using the "algebraic" properties (i.e., its transitive, reflexive and symmetric properties). The matching algorithms used by K$^{n}{}_{A}$c are described in detail in Chapter 5.

For the match between EVENT-1 and TAKE-A-TRIP-AND-GET-PAID (see Figures 10 and 11 in the previous chapter), for instance, the match proved *comparable* and resulted in the following field matches:[5]

[5]For brevity, reflexive constraints (e.g., $A = A$) are not shown and only one of each pair of symmetric constraints is included.

GENERALIZATIONS:

Sets match completely.

They share: { EVENT }.

SPECIALIZATIONS:

Sets trivially match.

PARTS:

Sets partially match (rating 0.333); 3.6% match likelihood.

They share: { TAKE_A_TRIP }.

{ ISSUE_TRAVEL_AUTHORIZATION } appears only in first entity.

{ GET_REIMBURSED } appears only in second entity.

PART-OF:

Sets trivially match.

ATTRIBUTE-NAMES:

Sets partially match (rating 0.250); 0.2% match likelihood.

They share: { DESTINATIONS }.

{ ACTOR } appears only in first entity.

{ COST TRAVELER } appear only in second entity.

CONSTRAINTS:

Constraints were CONSISTENT.

Match rating 0.048;

They share: (DESTINATIONS = TAKE_A_TRIP.DESTINATIONS)

Additional constraints in second entity:

```
  (ISSUE_TRAVEL_AUTHORIZATION BEFORE GET_REIMBURSED)
  (TAKE_A_TRIP BEFORE GET_REIMBURSED)
  (ACTOR = GET_REIMBURSED.RECIPIENT)
  (ACTOR = TRAVELER)
  (COST = GET_REIMBURSED.AMOUNT)
  (COST = TAKE_A_TRIP.COST)
  (GET_REIMBURSED.AMOUNT = TAKE_A_TRIP.COST)
  (GET_REIMBURSED.RECIPIENT = ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
  (GET_REIMBURSED.RECIPIENT = TAKE_A_TRIP.TRAVELER)
  (TRAVELER = GET_REIMBURSED.RECIPIENT)
  (TRAVELER = ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
  (TRAVELER = TAKE_A_TRIP.TRAVELER)
Additional constraints in first entity:
  (ISSUE_TRAVEL_AUTHORIZATION BEFORE GET_REIMBURSED)
  (ISSUE_TRAVEL_AUTHORIZATION BEFORE TAKE_A_TRIP)
  (DESTINATION OUTSIDE STATE)
  (ACTOR = GET_REIMBURSED.RECIPIENT)
  (ACTOR = ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
  (ACTOR = TAKE_A_TRIP.TRAVELER)
  (ACTOR = TRAVELER)
  (ISSUE_TRAVEL_AUTHORIZATION.ISSUEE = GET_REIMBURSED.RECIPIENT)
  (ISSUE_TRAVEL_AUTHORIZATION.ISSUEE = TAKE_A_TRIP.TRAVELER)
  (TRAVELER = ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)

ASSOCIATED-OBJECTS:
Sets trivially match.
```

§3. Evaluating Matches

In most recognition systems, the quality of a match is determined by how much (in relative or in absolute terms) two structures have in common. For a knowledge acquisition system, this is insufficient. Given $K^n{}_Ac$'s model of knowledge acquisition, information provided by the domain expert results in modifications to existing knowledge. This implies that differences between the expert's descriptions and those in the knowledge base are not only possible, but crucial to the acquisition process.

Therefore, in addition to the degree of fit, another metric for evaluating matches is necessary. This is accomplished in $K^n{}_Ac$ by providing a context-dependent matching process. Since differences recognized by the matcher imply modifications to one (or more) of the entity descriptions, $K^n{}_Ac$ provides a context for evaluating such matches by attempting to anticipate these modifications. Thus, differences between the descriptions do not necessarily detract from a match; only *unanticipated* differences do.

Consider the match results shown in the previous section. The match between the task description provided by the expert and the description of TAKE-A-TRIP-AND-GET-PAID is far from perfect. Because of these discrepancies, the context independent rating assigned to this match (as described in Chapter 5 Section §2.1) is only 0.271. However, many of the modifications implied by the differences are anticipated by $K^n{}_Ac$'s expectations. Consider the "extra" step (ISSUE-TRAVEL-AUTHORIZATION) in the expert's description of the task. This difference implies the modification:

```
MOD1516: ADD Issue_travel_authorization to the
                Parts field of Take_a_trip_and_get_paid
```

This modification is anticipated by expectations resulting from, respectively, the small number of steps in the existing event description, an unsatisfied precondition in the GET-REIMBURSED step, and the proximity (in the knowledge base) of TAKE-A-TRIP-AND-GET-PAID to the stated topic of discourse (TRAVEL):

```
Exp145:  Expecting (certainty 0.230):
   MOD:  ADD ?New-part<is-a-knac-structure-p> to the
             Parts field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_S2.

Exp83:  Expecting (certainty 0.307):
  MOD:  ADD ?New-step<is-an-event-p> to the
            Parts field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S3.

Exp21:  Expecting (certainty 0.100):
  MOD:  ?ACTION ?Value to/from the
               ?Field field of Take_a_trip_and_get_paid
        Derived from Travel and H_D1.
```

The details of both phases of this match evaluation process are presented in Chapter 5.

§4. Modifying the Knowledge Base

Once the best matches for the expert's descriptions have been determined, the modifications implied by the differences between the expert's description and the

existing description may be used to update the knowledge base. For descriptions failing to match any existing entity descriptions, new entities are generated. Depending on the degree of autonomy given to the K^nAc system, each match may be verified with the expert, each modification may be verified, only unexpected modifications (from selected matches) may be checked, only modifications that add information may be made, or the system may be permitted to make all the implied changes.

Given the match between EVENT-1 and TAKE-A-TRIP-AND-GET-PAID, for example, the modifications include:

```
MOD1516: ADD Issue_travel_authorization to the
             Parts field of Take_a_trip_and_get_paid

MOD1518: ADD Actor to the
             Attribute-names field of Take_a_trip_and_get_paid

MOD1521: ADD (Before issue_travel_authorization get_reimbursed)
             to the Constraints field of Take_a_trip_and_get_paid
```

Rather than destructively changing the actual structure descriptions contained in the knowledge base, K^nAc uses a "context" mechanism to create modified copies of the original structures. These contexts are organized as a tree, which permits the inheritance of descriptions from previous (parent) contexts and the existence of alternative (sibling) interpretations.

A context mechanism is provided for two reasons. First, the assimilation of the expert's information into the existing knowledge base is a heuristic process subject to incorrect interpretations. By establishing a new context for each

discourse frame, backtracking to an earlier part of the acquisition dialog merely requires "popping" the context stack to the appropriate point in the discourse.

The second advantage of having a context mechanism is the ability to maintain multiple views of the knowledge base. If the information provided by the domain expert is intended to correct or augment the existing knowledge in a global fashion, the resulting changes should apply to all future references to the modified structures. However, if the information is provided in order to customize the descriptions for a particular individual (or role or department, etc.), then the changes should only be made within that context. Maintaining the original and the modified structure descriptions in separate contexts permits different views of the knowledge base.[6]

§5. Generating Expectations

As was seen earlier in this chapter, $K^{n}{}_{A}c$ attempts to anticipate modifications to an existing knowledge base in order to better interpret the domain expert's descriptions. These expectations are generated using a set of heuristics that capture some of the knowledge engineer's expertise. These heuristics are based on the state of the knowledge base, the discourse, recent modifications, and the knowledge acquisition strategies being used.

The heuristics are, in essence, production rules that generate expectations when they run. The "trigger" for each rule depends on the class of heuristic. *State* heuristics are activated by the recognition of inconsistencies or holes in

[6] Assimilating the modified descriptions as *specializations* of the original one is another viable approach. However, backtracking, a potentially important part of the knowledge acquisition process, would not be quite as straightforward.

the knowledge base descriptions. *Discourse* heuristics rely on explicit discourse queues from the discourse manager (such as "topic" information) and on references to known entities. *Modification* heuristics are triggered by changes made to the knowledge base. *Acquisition strategy* heuristics[7] depend on the state of the knowledge acquisition session. A complete explanation of how expectations are generated from these heuristics and of how they are maintained is presented in Chapter 6.

§6. Interfaces to the System

$K^n{}_Ac$ must be able to access the expert system's knowledge base in order to create, delete, read and modify the structures contained therein. In order to minimize its dependence on a particular expert system, most of $K^n{}_Ac$'s mechanisms were designed to be extremely general. Only the actual structure access function code and the state "completeness" and "consistency" code must be customized for different knowledge representations. In addition, the set of knowledge acquisition heuristics may require modification for use with a different expert system.

Similarly, $K^n{}_Ac$ should not be restricted to a particular set of tools with which it may interact with the domain expert. The system requires a set of entity descriptions (in the language of the knowledge base on which it is working) and "discourse" information from the interface. While the examples presented herein have assumed a natural language frontend and discourse manager, such as POISE used within its "help" facility (see [DW83], [McD83] and [WPML84]), nothing in $K^n{}_Ac$ depends on this type of interface. A graphic procedure specification and display tool (such as the one designed for POISE's knowledge structures) could

[7] These are not currently used.

provide another type of frontend to the $K^n{}_Ac$ system.

Chapter 5

Matching for Knowledge Acquisition

As part of the knowledge acquisition task, information presented by the domain expert must be compared with that already in the knowledge base. This may be viewed as an interpretation task, but presents certain challenges unique to knowledge acquisition. Specifically:

1. **Generality:** Since the $K^{n}{}_{A}c$ system is intended to operate with various knowledge bases, each containing a variety of knowledge structures, the matching mechanism needs to either be sufficiently general or must be easy to customize.

2. **Varying matching criteria:** The criteria for determining whether two structures match changes with the context in which the match is being made. The same results may be judged acceptable or unacceptable under different circumstances.

3. **Using mismatches:** Typically, interpretation systems need only determine whether or not structures match and are not concerned with how matches fail. Some systems utilize these mismatches to backtrack to a "correct" interpretation. However, for knowledge acquisition, mismatches do not always indicate failure; they may indicate required knowledge base modifications.

This chapter examines how these requirements affect the matching process and presents $K^n{}_Ac$'s solutions to these additional demands. Specifically, Section §1. describes the generic matching techniques used by $K^n{}_Ac$ and how they may be augmented for particular knowledge structures; Section §2. presents the two-pass technique $K^n{}_Ac$ uses to evaluate the results of these matches.

§1. Match Techniques

When various frame-based knowledge representations were examined, such as those found in POISE, Knowledge Craft, ART and KEE,[1] several types of structures proved to be ubiquitous. The two most common structures were *sets of elements* and *collections of constraints*. In the representation used by $K^n{}_Ac$ for example, the *steps* field in an *event* description contains a set of the names of the constituent events. Similarly, the *part-of*, *specializations*, *generalizations* and *attributes*[2] fields each contain a set of elements. Though the semantics of the different fields vary, the matching algorithm presented in Section §1.1 may be used to syntactically compare each of these fields.

On the other hand, the *constraints*, *temporal-relationships*, *preconditions*, *postconditions* and *attribute definitions* fields of *events* each contain a collection of constraints. Again, though their semantics vary, each may be compared as described in Section §1.2.

[1]Knowledge Craft, ART and KEE are registered trademarks of Carnegie Group Inc., Inference, and IntelliCorp, respectively.

[2]Currently, the *attributes* field contains both the names of the event attributes and their definitions. The attribute names are handled by the set-matching mechanism, while the attribute definitions are treated as "equal" constraints.

§*1.1* *Set Matching*

Perhaps the simplest means of comparing two sets of elements uses the (absolute or relative) size of the intersection of the sets as a metric of how well they match. A more sophisticated approach may consider the extent of the mismatch between the two sets (e.g., the *contrast model* in [Tve77]) to determine how "different" they are. A third consideration, unique to knowledge acquisition, is the possibility of the sets being modified so as to match.

If there exists a means for comparing a pair of elements of the sets,[3] the extent to which the sets match may be expressed as the ratio of the size of the intersection to the size of the union of the sets. This metric, known as Jaccard's coefficient, is but one of several approximately equivalent measures of similarity. (See [vR79], pp. 38–39.) Formally:

$$\text{match-rating} = \begin{cases} 1 & \text{if } \mathrm{Set}_1 = \mathrm{Set}_2 = \emptyset \\ \dfrac{|\mathrm{Set}_1 \cap \mathrm{Set}_2|}{|\mathrm{Set}_1 \cup \mathrm{Set}_2|} & \text{otherwise.} \end{cases}$$

Thus, the match-rating for the *steps* field of events TAKE-A-TRIP-AND-GET-PAID, which contains TAKE-A-TRIP and GET-REIMBURSED, and EVENT-1, which contains ISSUE-TRAVEL-AUTHORIZATION and TAKE-A-TRIP, is:

$$\begin{aligned} &\text{match-rating} \\ &\quad = \frac{|\{\text{TAKE-A-TRIP}\}|}{|\{\text{ISSUE-TRAVEL-AUTHORIZATION TAKE-A-TRIP GET-REIMBURSED}\}|} \\ &\quad = 1/3 \end{aligned}$$

In general, each of the two sets may contain elements not found in the other

[3]This may be non-trivial. If each element is itself a "structure" like the one that contains the aforementioned set, the comparison process must be applied recursively.

set. If the "extra" elements in the first set are added to the second set, the first set will subsume the second. In order for the two sets to completely match, however, the extra elements in each set must be added to the other.[4] For example, adding ISSUE-TRAVEL-AUTHORIZATION as a step in TAKE-A-TRIP-AND-GET-PAID and adding GET-REIMBURSED as a step in EVENT-1 would make the sets match.

The likelihood of such modifications depends on the particular structures being modified. When adding a step to an event, for example, the likelihood that the event will contain another step and that the additional step will be of the type specified must be considered. Thus, the expected magnitude of each set and the possible range of values for its elements must be taken into account.

To compare two sets, say Set_1 and Set_2, each of whose typical size and range of elements is known, the following notation is introduced:

$$\begin{aligned}
E_i &:= \text{Elements in Set}_i \\
Ex_i &:= \text{Elements exclusive to Set}_i \\
M_i &:= \text{Typical magnitude of Set}_i \\
R_i &:= \text{Range of an element of Set}_i \\
R_{i,j} &:= \text{Intersection of the ranges of} \\
&\qquad \text{Set}_i \text{ and Set}_j \equiv R_i \cap R_j \\
S_i &:= \text{Available slots in Set}_i \equiv M_i - |E_i|
\end{aligned}$$

The probability of all the *extra* elements in Set_1 fitting into the available positions in Set_2 is:

$$P(Ex_1 \text{ fits into } S_2)$$

$$= \prod_{n=1}^{|Ex_1|} P(Ex_1(n) \text{ fits into } S_2)$$

[4]This takes only the addition of elements to these sets into account and ignores the possibility of the deletion or replacement of elements.

$$= \prod_{n=1}^{|Ex_1|} (1 - P(Ex_1(n) \text{ doesn't fit } \textit{any} \text{ remaining slot in Set}_2))$$

$$= \prod_{n=1}^{|Ex_1|} \left(1 - \left(P(Ex_1(n) \text{ doesn't fit } \textit{each} \text{ remaining slot}\right)^{(\text{\# of remaining slots})}\right)$$

If the range R_1 or R_2 is empty, the probability of an element from Set_1 fitting a slot in Set_2 is 0. Otherwise:

$$\begin{aligned}
P(Ex_1(n) &\text{ fits into } S_2(m)) \\
&= P(\text{the Set}_1 \text{ element is in the intersecting range}) \times \\
&\quad P(\text{the Set}_2 \text{ slot is in the intersecting range}) \times \\
&\quad P(\text{the element matches the slot, if both are in the intersection}) \\
&= P(Ex_1(n) \in R_{1,2}) \times P(S_2(m) \in R_{1,2}) \times \\
&\quad P((Ex_1(n) = S_2(m)) \mid \{Ex_1(n), S_2(m)\} \in R_{1,2}) \\
&= \frac{|R_{1,2}|}{|R_1|} \times \frac{|R_{1,2}|}{|R_2|} \times \frac{1}{|R_{1,2}|} \\
&= \frac{|R_{1,2}|}{|R_1| \times |R_2|}
\end{aligned}$$

If a potential match is to succeed, all of the extra elements in each set must be added to the other set. Therefore, the probability of a match is:

$$\begin{aligned}
P(\textit{match}) &= P(Ex_1 \text{ fits into } S_2) \times P(Ex_2 \text{ fits into } S_1) \\
&= \prod_{n=1}^{|Ex_1|} \left(1 - \left(1 - \frac{|R_{1,2}|}{|R_1| \times |R_2|}\right)^{|S_2|-n+1}\right) \times \\
&\quad \prod_{n=1}^{|Ex_2|} \left(1 - \left(1 - \frac{|R_{1,2}|}{|R_1| \times |R_2|}\right)^{|S_1|-n+1}\right)
\end{aligned}$$

For the sets described above, the following values may be used:[5]

$$
\begin{aligned}
E_1 &:= \{\text{ISSUE-TRAVEL-AUTHORIZATION TAKE-A-TRIP}\}\\
E_2 &:= \{\text{TAKE-A-TRIP GET-REIMBURSED}\}\\
Ex_1 &:= \text{ISSUE-TRAVEL-AUTHORIZATION}\\
Ex_2 &:= \text{GET-REIMBURSED}\\
M_i &:= 4\\
R_i &:= \text{list of travel-related entities}\\
R_{i,j} &:= \text{list of travel-related entities}\\
S_1 &:= 2\\
S_2 &:= 2
\end{aligned}
$$

Thus,

$$
\begin{aligned}
P(match) &= \prod_{n=1}^{1}\left(1-\left(1-\frac{10}{10\times 10}\right)^{2-n+1}\right)\times\\
&\quad \prod_{n=1}^{1}\left(1-\left(1-\frac{10}{10\times 10}\right)^{2-n+1}\right)\\
&= \left(1-(.9)^2\right)\times\left(1-(.9)^2\right)\\
&\approx 0.036
\end{aligned}
$$

Because these probabilities reflect the likelihood to both the entity descriptions provided by the domain expert and the ones already in the knowledge base, the values tend to be quite low. If only modifications to the knowledge base structures are considered, so that the modified knowledge base structures subsume, but need not exactly match, those provided by the expert, significantly higher match probabilities result.

[5]For each field in a structure, the *range* for elements in the field and the typical *size* of the field may either be specified as *facets* of that field or may be inherited along specified links.

§1.2 *Constraint Matching*

While comparing (or combining) arbitrarily constrained sets of entities is a difficult problem often requiring substantial domain knowledge, two sets whose members are (pairwise) related by a common relation may be compared in a purely mechanical fashion.[6] This section describes a mechanism for comparing such constrained sets; the mechanism is independent of the particular relation, requiring only a description of the algebraic properties of the relation.

Consider, for example, the *temporal relationships* field of an *event*. It contains pairs of (sub)events related by temporal ordering relations, (e.g., BEFORE, AFTER, DURING). In addition to the explicitly stated constraints, others may be inferred from the semantics of the particular relation; if *event-1* occurs before *event-2* and *event-2* occurs before *event-3*, then the constraint stating *event-1* occurs before *event-3* may be inferred.

The comparison of two sets of constraints is greatly simplified if all the implicit constraints are made explicit. In order to perform this constraint propagation, the only required description of the relation is whether it is **reflexive**, **symmetric** and/or **transitive**, and whether any of these properties are prohibited. For a set of entities (e.g., a, b, c), the term **non-reflexive** may be defined to describe a relation such that $(a \rightarrow a)$, meaning the relation from a to a, **cannot** be true. Similarly, **non-symmetric** and **non-transitive** are defined to prohibit $((a \rightarrow b) \wedge (b \rightarrow a))$ and $((a \rightarrow b) \wedge (b \rightarrow c) \wedge (a \rightarrow c))$, respectively. Note that the **non-symmetric** property differs slightly from the more common **anti-symmetric** property which states $((a \rightarrow b) \wedge (b \rightarrow a)) \Rightarrow (a = b)$. The temporal relation

[6]The current system checks each relation separately; it does not handle interaction between different relations, though it is able to combine relations with their inverses. For instance, *before* and *after* constraints are handled together.

before, for example, is non-reflexive, non-symmetric and transitive; EVENT-A cannot occur before itself, EVENT-A cannot occur before EVENT-B if EVENT-B occurs before EVENT-A, and if EVENT-A occurs before EVENT-B and EVENT-B occurs before EVENT-C, then EVENT-A occurs before EVENT-C.

Propagating Constraints

As was seen with *temporal relationships*, not all of the valid relationships may have been expressed explicitly. In order to simplify the determination of consistency and the combining of these constrained sets, all implicit relations are made explicit by propagating the constraint. If $E_{i,j}$ indicates that the relation ($\mathrm{var}_i \rightarrow \mathrm{var}_j$) holds, the reflexive, symmetric and transitive closures may be generated as follows:

1. If the constraint is **reflexive**, $\forall(i)E_{i,i}$;
2. If the constraint is **symmetric**, $\forall(i,j)E_{i,j} \Rightarrow E_{j,i}$;
3. If the constraint is **transitive**, $\forall(i,j,k)E_{i,j} \wedge E_{j,k} \Rightarrow E_{i,k}$.

By representing the constrained variables as a $n \times n$ matrix C, where n is the number of variables, the propagation of reflexive and symmetric constraints is implemented in a straightforward manner. Propagating transitive constraints is accomplished by generating a matrix M, such that:

$$M = C + C^2 + \cdots + C^{n-1}.$$

Matrix M represents the transitive-closure of the constraint over the variables.[7] (See [AAM81], pp. 196–197.)

[7]Since only whether or not a constraint holds for a pair of variables is of interest, the entries in the matrices may be restricted to boolean values. Thus, the "+" operator may be treated as a logical-or.

Consider the following sets of relations constrained by the temporal relationship *before* ($<$):

$$\text{Relations 1:} \begin{cases} a < b \\ b < c \\ c < d \end{cases} \qquad \text{Relations 2:} \begin{cases} a < d \\ d < b \end{cases}.$$

The resulting matrices are:

$$\text{Matrix 1:} \left\{ \begin{array}{c|cccc} & a & b & c & d \\ \hline a & - & \checkmark & - & - \\ b & - & - & \checkmark & - \\ c & - & - & - & \checkmark \\ d & - & - & - & - \end{array} \right. \qquad \text{Matrix 2:} \left\{ \begin{array}{c|ccc} & a & b & d \\ \hline a & - & - & \checkmark \\ b & - & - & - \\ d & - & \checkmark & - \end{array} \right.$$

After propagating the *before* constraint, which just requires generating the transitive-closure of the above matrix since *before* is only **transitive**, the matrices contain:

$$\text{Matrix-tc 1:} \left\{ \begin{array}{c|cccc} & a & b & c & d \\ \hline a & - & \checkmark & \checkmark & \checkmark \\ b & - & - & \checkmark & \checkmark \\ c & - & - & - & \checkmark \\ d & - & - & - & - \end{array} \right. \qquad \text{Matrix-tc 2:} \left\{ \begin{array}{c|ccc} & a & b & d \\ \hline a & - & \checkmark & \checkmark \\ b & - & - & - \\ d & - & \checkmark & - \end{array} \right.$$

Once the constraint has been completely propagated, violations of non-reflexive, non-symmetric and non-transitive constraints may be checked. Specifically, inconsistencies exist if:

1. The constraint is **non-reflexive** $\wedge$ $(\exists\,(i) \mid E_{i,i})$;

2. The constraint is **non-symmetric**
   $\wedge (\exists\,(i,j) \mid ((i \neq j) \wedge E_{i,j} \wedge E_{j,i}))$;

3. The constraint is **non-transitive**
$\wedge(\exists(i,j,k) \mid ((i \neq j \neq k) \wedge E_{i,j} \wedge E_{j,k} \wedge E_{i,k}))$.

Thus, neither set of relations shown above is inconsistent, since neither the **non-reflexive** nor the **non-symmetric** properties have been violated.

After applying the above procedures to each set (to assure internal consistency), the two sets may be compared by combining the sets and then applying the above procedure to the combined set. A matrix representing the combined sets be may generated by expanding the matrix for each set to a $|V_1 \cup V_2| \times |V_1 \cup V_2|$ matrix, where V_1 and V_2 are the variables in each of the original sets.

Expanding and combining the above matrices produces:

$$\text{Combined-matrix:} \left\{ \begin{array}{c|cccc} & a & b & c & d \\ \hline a & - & \checkmark & \checkmark & \checkmark \\ b & - & - & \checkmark & \checkmark \\ c & - & - & - & \checkmark \\ d & - & \checkmark & - & - \end{array} \right.$$

The transitive-closure of the combined relations is:

$$\text{Combined-matrix-tc:} \left\{ \begin{array}{c|cccc} & a & b & c & d \\ \hline a & - & \checkmark & \checkmark & \checkmark \\ b & - & \checkmark & \checkmark & \checkmark \\ c & - & \checkmark & \checkmark & \checkmark \\ d & - & \checkmark & \checkmark & \checkmark \end{array} \right.$$

This resulting set is, therefore, inconsistent. Both the **non-reflexive** and the **non-symmetric** properties are violated (by $\{b < b, c < c, d < d\}$ and $\{b < c < b,$

$b < d < b$, $c < d < c$ } respectively). Note that the relations "responsible" for the inconsistencies have not been detected, merely those inconsistent relations that resulted.

If the second set of relations is changed so that

$$\text{Relations 1:} \begin{cases} a < b \\ b < c \\ c < d \end{cases} \qquad \text{Relations 2:} \begin{cases} a < d \\ b < d \end{cases}$$

the following (consistent) matrix would result:

$$\text{Combined-matrix-tc:} \left\{ \begin{array}{c|cccc} & a & b & c & d \\ \hline a & - & \checkmark & \checkmark & \checkmark \\ b & - & - & \checkmark & \checkmark \\ c & - & - & - & \checkmark \\ d & - & - & - & - \end{array} \right.$$

Compacting Constraints

To simplify the processing of a set of constrained variables, information implicitly contained in the constraint was made explicit by generating the appropriate closures of the set. The results, therefore, may contain information that is redundant, given the constraint. To minimize the modifications made to the knowledge base or the information supplied to the domain expert, these results may be expressed more succinctly by removing this inherent information; this is done by generating the appropriate "opensures".[8]

[8] The term "opensure" is being used to describe a set that is conceptually opposite to that of a "closure". The correct pronunciation is obtained by placing one's tongue firmly in one's cheek.

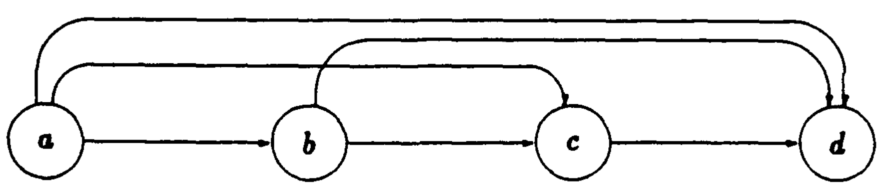
Let us define the opensure (0) of a set of constrained variables (C) as a subset of C with the smallest possible cardinality such that Closure(0) = C. More than one such subset may exist. If an ordering is imposed on the variables in C, e.g., lexicographic, the following algorithm will produce a unique solution.

The algorithm that generates the opensure must assure that:

1. If the constraint is **reflexive**, $\forall(i)\ \overline{E}_{i,i}$;
2. If the constraint is **symmetric**, $\forall(i,j) \mid (i<j)\ \overline{E}_{i,j}$;
3. If the constraint is **transitive**,
   $\forall(i,j,k) \mid (i \neq j \wedge j \neq k \wedge E_{i,j} \wedge E_{j,k})\ \overline{E}_{i,k}$.

Again, by making use of the matrix representation of the constrained set to generate the opensure, the removal of information resulting from reflexive and symmetric properties is straightforward: remove elements from the diagonal and the lower-left half of the matrix, respectively. This information can be "recovered" from the remaining matrix and a knowledge of the constraint's properties. The removal of the information resulting from transitive closure is a more interesting problem.

Perhaps the easiest way to understand the solution is to recast the information as a directed graph, where element $E_{i,j}$ now indicates a directed arc from $node_i$ to $node_j$. The above matrix may be represented as the following graph:



If the constraint in question is non-reflexive, there can be no nodes in the graph that point directly to themselves (i.e., there are no cycles containing exactly one node). If it is non-symmetric, there can be no cycles containing two (or more) nodes. If the constraint is either reflexive or symmetric, the above opensure operations will remove cycles of one or more than one nodes. If the constraint is neither reflexive nor non-reflexive (or it is neither symmetric nor non-symmetric), the graph may contain cycles. First, the case of an acyclic, directed graph is examined and then the handling of such cycles will be discussed.

The following algorithm generates the transitive opensure of the acyclic constraint set, C:

1. Find the set of minimum nodes (those which are pointed to by no other nodes) in C; call this set M and its members $m_1 \ldots m_n$.

2. For each m, find the set of nodes $\{n_1 \ldots n_n\}$ in C such that $m \rightarrow n$. This set, N, is the set of nodes to which m connects.

3. For all $n_i, n_j \in N$, select those n_j such that $n_i \rightarrow n_j$ does not hold. This set of n_j's, call it P, is the set of "closest" nodes to m. Include $m \rightarrow p$, for all $p \in P$, in the opensure and recursively apply steps 2 and 3 from all nodes in P.

Since there are no cycles in the above graph, this algorithm may be used. The set of minimum nodes in the graph is $\{a\}$. The set of nodes to which a is connected is $\{b, c, d\}$. Of this set, only b is not pointed to by another member of the set; thus, the set of nodes "closest" to a is $\{b\}$. Applying this procedure to b, produces $\{c\}$; for c, $\{d\}$. Thus, the resulting graph is:

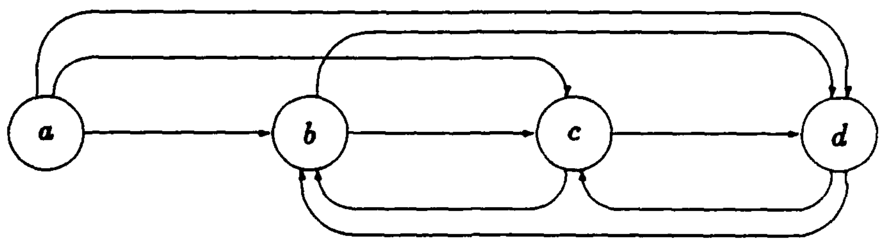
If the graph contains cycles, it may be preprocessed to remove the cycles and then processed as described above. Cycles containing exactly one may be handled trivially: these constraints, located on the diagonal of the matrix, are removed from the set and placed directly into the opensure unless there is another (multi-node) cycle that contains this node.

Cycles containing exactly two nodes may be handled by designating one arc of the pair as "forward" and the other as "backward". (Since a (possibly arbitrary) ordering has been assigned to the nodes, this designation is simple.) Removing the set of "backward" arcs removes all two-node cycles. The (now) acyclic graph may be processed as above, the (also acyclic) graph comprised of the set of "backward" arcs processed in the same way, and the two results combined. It is readily seen that any cycles containing more than two nodes are reduced to a set of two-node cycles in the transitive closure and the above method is therefore applicable.

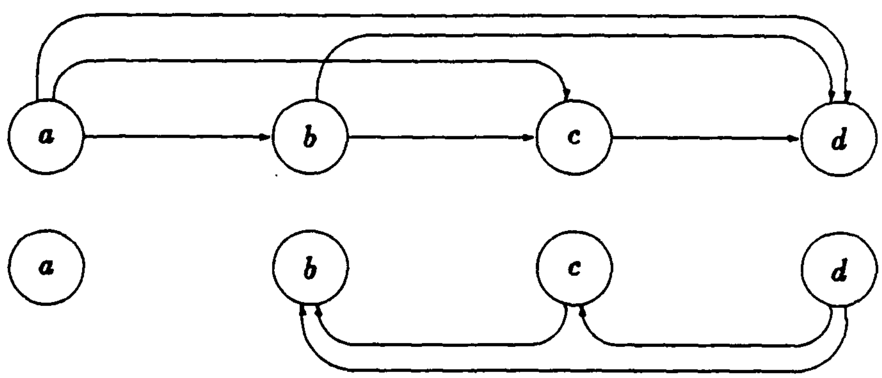
Changing the constraint to *not-after* ($\leq$), i.e., *before* or *at-the-same-time* (which permits, but does not require, symmetric relations), and modifying the second set of relations such that:

$$\text{Relations 1:}\begin{cases} a \leq b \\ b \leq c \\ c \leq d \end{cases} \qquad \text{Relations 2:}\begin{cases} a \leq d \\ b \leq d \\ d \leq b \end{cases}$$

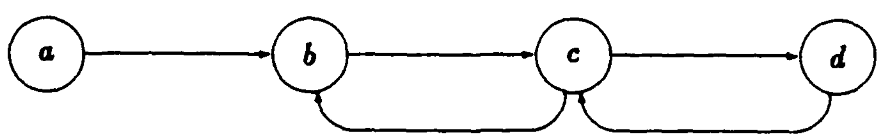
produces the closure:



The symmetric opensure divides the symmetric pairs, producing:



The transitive opensure then reduces each graph separately and then combines them to produce:



Merging Sets of Constraints

Since the goal of knowledge acquisition is the modification of the existing knowledge base to include new information, an existing constrained set may need to be modified to include additional relations. If both the existing set and the set of additions are consistent, both internally and with each other, the new relations may simply be added to the existing ones. However, this may not produce a minimal set of relations and may involve adding more information than is necessary.

Consider two sets of relations, S_1 and S_2, whose members are related by a transitive constraint, such as "before":

$$\begin{aligned} S_1 &= \{(a \rightarrow b)\} \\ S_2 &= \{(a \rightarrow c)(b \rightarrow c)\} \end{aligned}$$

Note that adding S_2 to S_1 would add superfluous information; adding $(b \rightarrow c)$ alone would suffice.

Determining the minimal set of relations that need to be added is, in general, a non-trivial problem. Simply adding the "new" relations, or even the opensure of these relations, is not necessarily the minimal solution. The correct solution requires generating the set of "additional relations" while keeping in mind the set to which it will be added.

In order to minimize this (additional) set, the opensure of the propagated relations that contains a maximum number of those already in S_1 must be found. From the viewpoint of the directed graph representation, the opensure must be generated while preserving a specified set of links. Since the only part of the opensure procedure that provides any leeway as to what relations are kept

or discarded is the removal of "symmetric" information, it is there that the algorithm must be modified. Given a symmetric pair of relations (e.g., $\{(a \rightarrow b)(b \rightarrow a)\}$, and a set of relations to preserve, the selection of which to remove may no longer be arbitrary (or lexicographic); the appropriate relation must be selected. If one relation of the pair is in the set of relations to preserve, the other should be removed; if both are in the set, either may be removed.

If neither relation need be preserved, the choice may still not be arbitrary; the choice may affect which other relations get removed. Consider the cycle formed by the three symmetric pairs of relations resulting from $a = b = c$, namely $\{(a = b)(b = a)(a = c)(c = a)(b = c)(c = b)\}$. (Ignore the reflexive relations (e.g., $(a = a)$) for now.) If $\{(a = b)(b = c)(a = c)\}$ are kept, $(a = c)$ will be removed when the transitive opensure is generated. If, instead $\{(b = a)(b = c)(a = c)\}$ are chosen, $(a = c)$ will remain and $(b = c)$ will be removed. Therefore, in selecting between $(a = b)$ and $(b = a)$, any relations involving a or b that should be preserved must be considered.

In the case described above, selection of the wrong relation will result in a cycle. To avoid this, the relation $(a \rightarrow b)$ is not selected if, for some x, both $(x \rightarrow a)$ and $(b \rightarrow x)$ must be preserved. If both $(x \rightarrow a)$ and $(x \rightarrow b)$ (or both $(a \rightarrow x)$ and $(b \rightarrow x)$) are to be kept, either $(a \rightarrow b)$ or $(b \rightarrow a)$ may be selected, but in either case, one of the two desired relations will be eliminated. If none of the above situations apply, there may be, at most, one relation adjacent to $(a \rightarrow b)$ that is to be preserved. If it involves a (i.e., either $(x \rightarrow a)$ or $(a \rightarrow x)$), $(a \rightarrow b)$ may be discarded, and vice versa for b.

Rating Constraint Matches

For each field of a structure containing constraints, the constraints are grouped by relationship (as described above) and the match for each relationship is rated. The values for all the relationships are averaged and this is used as the rating for that field. For each relationship in the field, if the two sets of constraints are found to be consistent, the match is rated as follows:

$$\text{Rating} = \frac{|\text{Constraints in common}|}{|\text{Resulting constraints}|}$$

Consider the match between the *constraints*[9] fields of EVENT-1 and TAKE-A-TRIP-AND-GET-PAID. EVENT-1 contains the constraints:

```
(EQUAL ACTOR ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
(EQUAL DESTINATIONS TAKE_A_TRIP.DESTINATIONS)
(EQUAL ISSUE_TRAVEL_AUTHORIZATION.ISSUEE TAKE_A_TRIP.TRAVELER)
(OUTSIDE DESTINATION STATE)
(BEFORE ISSUE_TRAVEL_AUTHORIZATION TAKE_A_TRIP)
```

while TAKE-A-TRIP-AND-GET-PAID contains:

```
(EQUAL COST TAKE_A_TRIP.COST)
(EQUAL DESTINATIONS TAKE_A_TRIP.DESTINATIONS)
(EQUAL TRAVELER TAKE_A_TRIP.TRAVELER)
```

[9] As part of the initialisation of structures, the attribute definitions (where provided), the temporal relationships (for events) and the constraints specified in the *constraints* field are merged. This allows consistency to be maintained for all the constraints specified for the structure.

```
(EQUAL TAKE_A_TRIP.TRAVELER GET_REIMBURSED.RECIPIENT)
(EQUAL TAKE_A_TRIP.COST GET_REIMBURSED.AMOUNT)
(BEFORE TAKE_A_TRIP GET_REIMBURSED)
```

The constraints are grouped according to their relations (i.e., *equal*, *outside* and *before*) and each group is rated separately. Out of seven *equal* constraints, there is one constraint in common, (EQUAL DESTINATIONS TAKE-A-TRIP.DESTINATIONS). Thus, the rating for the *equal* constraints is 1/7. Since neither the *before* nor the *outside* relations share any constraints, the average over the three relations is about 0.048.

§*1.3 Structure-specific Matching*

Consider now how two *event* structures may be compared. Recall that *events* consist of fields containing generalizations and specializations of the event, events that comprise and contain this event, temporal relations between sub-events, attributes, constraints on these attributes, associated objects, and pre- and post-conditions. We may compare the two events on a field-by-field basis using the general matching techniques described in the previous section. The *generalizations*, *specializations*, *parts*, *part-of*, *attribute* names and *associated-objects* fields may be treated as sets of entities and compared with the generic set comparison technique of Section §1.1. The *constraints*, *temporal-relationships*, *attribute* definitions and *pre-* and *post-conditions* are sets of constraints and can be handled by the constraint comparison mechanism of Section §1.2.

In addition to applying these generic matching techniques to specific fields of a particular type of structure, the matching procedure for each structure type may be customized in two ways. First, the way in which the results of matching

individual fields are combined (as described in Section §2.) and the heuristics used to anticipate modifications may vary with the type of structure; second, structure-specific matching techniques may be added as needed.

When matching *events*, for instance, an implicit consumer/producer relationship among its steps may contribute to an evaluation of the structure's completeness. That is, each step *may* require resources made available by earlier steps in the *event*. Examining the *pre-* and *post-conditions* of each step can reveal such dependencies. If the *pre-condition* of an *event* step cannot be determined to be satisfied by the *post-conditions* of any earlier steps nor contained as a *pre-condition* of the *event* itself, then the possibility of an "unsatisfied precondition" is raised. This condition allows potential modifications to be inferred as explained in Chapter 6 Section §1.1.

§2. Match Evaluation

In addition to the extent to which the expert's entity description matches (or differs from) an existing knowledge base entity, *how* they differ, and whether these differences are expected, must be considered to determine the acceptability of the match. Thus, the match evaluation procedure relies, in part, on previously generated modification expectations.

In this section, a two-pass match evaluator is described. The first pass uses just the "degree of fit" of the two entities. This is a typical evaluation technique and provides a quick, inexpensive means of pruning the set of possible matches. The second pass takes the context in which the comparison is made into account.

§2.1 *Context-Independent Match Evaluation*

Context-independent match evaluation provides a straight-forward way to determine the quality of the matches and to eliminate those that seem very unlikely. Using the results of the matching process described in Section §1., a numeric value is assigned to each match. Those failing to exceed a specified threshold are removed from consideration.[10]

The matching procedure determines if two entity descriptions (i.e., the expert's and one in the knowledge base) are *equal*, *related*, *comparable* or *different*. Matches deemed *equal* are assigned a maximum value of 1; those that are *different* receive a minimum value of 0. Entities which are recognized as being *related* matches are assigned a low value (currently, 0.2), to reflect the small chance that the expert's description is intended to match the target knowledge base entity, despite matching an entity related to it. For *comparable* matches, the evaluation consists of assigning a value to each match on a field-by-field basis and then averaging these values over all the fields.

The value used for each field match depends on the type of match involved. For *set matches*, the probability of a match (see Section §1.1) is the value used. For *constraint matches*, the consistency of the sets of constraints determines the value: inconsistent sets yield 0, while consistent sets use the match rating (see Section §1.2). Trivially consistent matches (i.e., fields that contain no information in either structure) are not assigned a value and are not averaged into the overall match value.

For the match between EVENT-1 and TAKE-A-TRIP-AND-GET-PAID, the non-

[10]Alternatively, a fixed number of the top-rated matches may be maintained, clustering techniques may be used, etc.

trivial field matches are rated as follows:

Field	Rating
generalizations	1.000
parts	0.036
attribute-names	0.002
constraints	0.048
Match Rating	0.271

The values thus obtained for the potential matches for EVENT-1 were:

Entity	Result	Rating
TAKE-A-TRIP-AND-GET-PAID	comparable	0.27
MAKE-A-RESERVATION	comparable	0.24
TRAVEL	comparable	0.24
KNAC-STRUCTURE	related	0.20
TAKE-A-TRIP	related	0.20
EVENT	related	0.20
DESTINATION	comparable	0.12
SOURCE	comparable	0.12
GO-SOMEWHERE	comparable	0.10
SEND-INFORMATION	comparable	0.09
PAY	comparable	0.05
TRAVELER	comparable	0.04
DRIVE	comparable	0.04
FLY	comparable	0.04
REQUEST-INFORMATION	comparable	0.03

§2.2 *Context-Dependent Match Evaluation*

Differences between a domain expert's description of an entity and a description already existing in a knowledge base may imply modifications to the existing description. By anticipating such modifications, these differences may contribute to, rather than detract from, a potential match. These expectations form a context in which the entity comparison may be evaluated.

The modifications implicit in the match results must first be made explicit. To resolve differences resulting from set mis-matches, the addition or removal of the "extra" elements to or from the appropriate fields is necessary. Similarly, for consistent constraint matches, additional constraints may have to be added or removed. These modifications are compared to the current set of expected modifications to better evaluate the matches that produced them.

In Frame 1, the potential modifications to TAKE-A-TRIP-AND-GET-PAID that would result if it were matched to EVENT-1 include:

```
MOD1516: ADD Issue_travel_authorization to the
             Parts field of Take_a_trip_and_get_paid

MOD1518: ADD Actor to the
             Attribute-names field of Take_a_trip_and_get_paid

MOD1521: ADD (Before issue_travel_authorization get_reimbursed)
             to the Constraints field
             of Take_a_trip_and_get_paid
```

Some of these modifications were anticipated. For MOD1516, for instance, the

following expectations were available:

```
Exp21:  Expecting (certainty 0.100):
  MOD:  ?ACTION ?Value to/from the
                ?Field field of Take_a_trip_and_get_paid
        Derived from Travel and H_D1.

Exp83:  Expecting (certainty 0.480):
  MOD:  ADD ?New-step<is-an-event-p> to the
            Parts field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S3.

Exp145:  Expecting (certainty 0.360):
   MOD:  ADD ?New-part<is-a-knac-structure-p> to the
             Parts field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_S2.
```

The context-dependent rating of a match is a measure of the extent to which the discrepancies between the descriptions can be accounted for. To determine this, the degree to which each modification implied by these discrepancies is expected must first be calculated. A pattern matcher is used to compare each modification to the modifications predicted by each of the currently expected modifications. Since a modification may be anticipated by multiple expectations, its "expectedness" can be described as a function of the certainties of these expectations. It is desirable that the expectedness of the modification 1) is at least as great as the certainty of the most certain applicable expectation, 2) increases as the number of applicable expectations increases, 3) is not dependent on the

order in which the expectations are considered, and 4) is in the range [0,1]. If C_i is the certainty of the ith expectation and E_i is the expectedness from the first i expectations, then the expectedness may be defined as:

$$E_i = \begin{cases} 0 & \text{if } i = 0 \\ E_{i-1} + C_i(1 - E_{i-1}) & \text{otherwise.} \end{cases}$$

Thus, for modification MOD1516 above, its expectedness is:

$$\begin{aligned} E_1 &= 0.360 \\ E_2 &= 0.360 + 0.480(1 - 0.360) \\ &\approx 0.667 \\ E_3 &\approx 0.667 + 0.100(1 - 0.667) \\ &\approx 0.700 \end{aligned}$$

The extent to which each field's match differences were anticipated is found by computing the average of the expectedness of the resulting modifications for that field. For the *parts* field match between EVENT-1 and TAKE-A-TRIP-AND-GET-PAID, for example, the resulting modifications are:

```
MOD1516: ADD Issue_travel_authorization to the
             Parts field of Take_a_trip_and_get_paid

MOD1517: REMOVE Get_reimbursed from the
                Parts field of Take_a_trip_and_get_paid
```

The expectedness for these modifications is 0.700 and 0.100, respectively. The expectedness for the *parts* field, therefore, is 0.400.

A slight refinement is added for constraint match fields. Since the constraints are grouped according to their relations before being compared (e.g., the "before" constraints are compared separately from the "equal" constraints), this additional structure may be used to evaluate the constraint fields expectedness. The average expectedness for each relation is found separately and these are then averaged to obtain expectedness for the constraint field. This prevents a large number of constraints for one relation from completely overwhelming the results of constraints from another.

For the *constraints* field, three relations were involved: *before*, *outside* and *equal*. The *before* constraints resulted in four modifications:

```
MOD1521: ADD (Before issue_travel_authorization get_reimbursed)
             to the Constraints field
             of Take_a_trip_and_get_paid

MOD1522: ADD (Before issue_travel_authorization take_a_trip)
             to the Constraints field
             of Take_a_trip_and_get_paid

MOD1540: REMOVE (Before issue_travel_authorization get_reimbursed)
                from the Constraints field
                of Take_a_trip_and_get_paid

MOD1541: REMOVE (Before take_a_trip get_reimbursed)
                from the Constraints field
                of Take_a_trip_and_get_paid
```

The expectedness of these modifications is 0.532, 0.100, 0.100 and 0.100, respectively. Thus, the average expectedness of the *before* modifications is 0.208. Similarly, the average expectedness for the both the *equal* and *outside* modifications is 0.100. Thus, the average expectedness for the *constraints* field is 0.136.

However, the expectedness of a field's modifications does not take the overall likelihood of the field match into account. The expectedness of each modification resulting from two poorly matching fields may be high, for instance, but the poor quality of the match should not be ignored. Therefore, the context-dependent field match rating is obtained by scaling the expectedness of the field's modifications by the context-independent rating of the field match. For this match, the context-dependent field ratings are:

Field	Rating
generalizations	1.000
parts	0.133
attribute-names	0.055
constraints	0.007
Match Rating	0.299

The dependent values for the best matches for EVENT-1 are:

Entity	Rating
TAKE-A-TRIP-AND-GET-PAID	0.30
MAKE-A-RESERVATION	0.17
TRAVEL	0.17

Thus, the ambiguity as to the best match for EVENT-1 resulting from the initial context-independent match process (resulting in ratings of 0.27, 0.24 and 0.24, respectively, for the above candidates) was largely resolved by examining

the degree to which the resulting modifications were expected.

Chapter 6

Generating and Managing Expectations

K^nA_c provides a context in which to interpret information provided by a domain expert by anticipating modifications to an existing knowledge base. These anticipated modifications are called *expectations* and are derived from K^nA_c's heuristic information about the knowledge acquisition process. (See Appendix B for a complete listing of K^nA_c's heuristics.) This chapter describes how these expectations are generated, how they are used to provide a context in which matches may be evaluated, and how they ranked and managed.

§1. Generating Expectations

In order to anticipate modifications to a knowledge base, K^nA_c contains a body of knowledge acquisition expertise. This expertise is in the form of a collection of heuristics culled from the analysis of knowledge acquisition protocols between a knowledge engineer and a domain expert. (See Chapter 3.)

Each heuristic contains, in addition to a *name* and *description*, a *type*, a *condition*, an *expectation* and a *specificity*. The current set of heuristics may be divided into four types. The first is based on the *state* of the knowledge, both that already contained in the knowledge base and new information provided by

the expert. The second type depends on *modifications* previously made to the existing knowledge base. The third type makes use of a model of the *discourse* process, while the fourth incorporates knowledge about teaching and learning *strategies*.

A heuristic's *condition* determines whether the heuristic is applicable in a particular context. The actual form of condition is determined by the heuristic's *type*. For instance, *modification* heuristics are triggered by changes to the knowledge base and therefore contain templates of modifications as their conditions. The template is compared, via a pattern matcher, to an actual modification; if the match succeeds, variables in the template are bound to values in the actual modification. These variables are used in the generation of the resulting *expectations*.

Consider the modification heuristic shown in Figure 19. The condition will only be satisfied when compared to a modification of type "add" and whose target (i.e., the structure being modified) is an "event".[1] If the condition is satisfied, the heuristic will generate an expectation anticipating the addition of a temporal relationship to the modified structure, **?event1**. This relationship will involve the newly added step, **step1**, and some other unspecified step, **step2**.

Each heuristic also assigns a *certainty* and an *effective-time-frame* to the expectations it generates. These are used to maintain a set of the most relevant expectations and are discussed further in Section §2.

The remainder of this section describes $K^{n}{}_{A}c$'s current set of knowledge acquisition heuristics and shows how they are used to generate expectations. No claims are made for the completeness or the optimality of this particular set of

[1] Pattern variables may be specified as either **?X** or **?(X** *pred***)**, where *pred* is a predicate on the variable that must be satisfied.

```
(create-heuristic h_m4a
   :description-string "Parts of Events are usually temporally
                        constrained after being introduced."
   :type 'modification
   :specificity 'specific
   :condition (make-modification
                 :action-type 'add
                 :field 'parts
                 :value '?step1
                 :target '?(event1 is-an-event-p))
   :expectation  (add-expectation
                    :cause 'h_m4a
                    :source '?event1
                    :modification
                      (make-modification
                        :action-type 'add
                        :field 'constraints
                        :value `(,?(temporal-relation
                                    is-a-temporal-relationship-p)
                                 ,'?step1
                                 ,?(step2 is-an-event-p))
                        :target '?event1)
                    :effective-time-frame 'fade
                    :certainty '?rating))
```

Figure 19: A Modification Heuristic

heuristics. Since it is expected that this set will be modified, both as a result of improved understanding of the knowledge acquisition process and through the addition of domain specific heuristics, K^nAc allows heuristics to be added (or removed) in a straight-forward way.

§*1.1* *State of the Knowledge*

Since one of the basic objectives of a knowledge acquisition system is to obtain complete and correct information, any partial or erroneous information is likely to be modified. The detection of missing or erroneous information must rely heavily on the underlying knowledge representation system. However, many of the shortcomings of a knowledge base, and the modifications they imply, may be expressed at a level that is not dependent on the particular knowledge representation. Thus, most of K^nAc's state-based heuristics rely on *some* knowledge base mechanism to determine incompleteness and inconsistencies, but are not dependent on any particular mechanism.

Consider a simple state heuristic relying on the detection of missing information:

Heuristic S6: ***Referenced entities should exist.***

If a description from the expert (or an existing entity description) contains a reference to an entity not already contained in the knowledge base, the addition of such an entity may be expected. The context in which the reference is made will often provide restrictions about the anticipated entity. For instance, if the unknown entity is referenced as a step of an event, then the facet information for this field, inherited (in this case) from the generic event description, requires that the new entity must also be an event. Where the type of the new structure cannot be determined from the available information, the most general type,

knac-structure, is used. For instance, TAKE-A-TRIP-AND-GET-PAID contains an attribute COST which is not a known knowledge base entity. Thus, an expectation is generated:

```
Exp82:  Expecting (certainty 0.800):
  MOD:  CREATE the Knac-structure Cost
        Derived from Take_a_trip_and_get_paid and H_S6.
```

A heuristic that relies more heavily on the completeness-checking ability of the underlying knowledge base system is:

Heuristic S2: *Fields with too few components will be augmented.*

This heuristic states that if information is detected to be missing, the addition of that information may be expected. Missing information may be detected in various ways. One simple approach compares the number of entries in a field of a knowledge structure with the field's expected cardinality. If the field is determined to contain too few values, additional values (of the appropriate type) will be expected. The expected field size may come, in order of specificity, from meta-information about a given field of a given entity, via inheritance from a generalization of the entity, from the default information for the type of the entity, or from an overall field default size. This size information may be static or determined dynamically by the system. An expectation generated by this heuristic is:

```
Exp146:  Expecting (certainty 0.360):
   MOD:  ADD ?New-part<is-a-knac-structure-p> to the
         Parts field of Take_a_trip_and_get_paid
```

```
Derived from Take_a_trip_and_get_paid and H_S2.
```

Whereas the quantitative analysis of the size of structure fields is a very generic means for inferring missing information, other methods may be more structure dependent. Missing steps in an event may be detected by assuming a consumer/producer relationship between steps of a task. In particular, a step in a plan may take place in order to produce something (a result, a resource, etc.) required by a subsequent step. Therefore, if the precondition of a step[2] is not satisfied by the previous steps in that event, it may be because a step responsible for satisfying that condition is missing. Thus, an unsatisfied pre-condition in a step of an event may result in the expectation of adding another (temporally constrained) step to that event. This is captured by:

Heuristic S3: *Unsatisfied step preconditions will be satisfied.*

The unsatisfied precondition in the GET-REIMBURSED step of the event TAKE-A-TRIP-AND-GET-PAID results in:

```
Exp84:  Expecting (certainty 0.480):
  MOD:  ADD ?New-step<is-an-event-p> to the
            Parts field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S3.

Exp85:  Expecting (certainty 0.480):
  MOD:  ADD (Before ?new-step<is-an-event-p> get_reimbursed)
            to the Constraints field
            of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S3.
```

[2]Remember, each step in an event is itself an event, (potentially) with pre- and post-conditions.

Actually, an event step with an unsatisfied precondition may imply several other problems as well: 1) there may be a missing or incorrect pre-condition in the (parent) event, 2) an earlier step may be missing or contain an incorrect post-condition, 3) the unsatisfied pre-condition may be extraneous or incorrect, or 4) the temporal ordering of the event steps may be wrong. Each of these explanations may also be used to generate expectations.

In addition to obtaining missing information, any incorrect information detected in the knowledge base is likely to be corrected. For instance, most knowledge base systems contain some mechanism for maintaining consistency. Perhaps the simplest such mechanism is some form of type-checking the structures in the knowledge base. One heuristic uses such errors to anticipate changes to values that violate range restrictions:

Heuristic S4: *Attribute values must be within specified ranges.*

Other heuristics generate expectations upon detecting cardinality conflicts and required values that are not present.

To generate state-based expectations appropriate for a particular time frame, the candidate match entities are checked for completeness and consistency. For the integration of the new information in the first time frame, the following state-based problems were discovered for the event TAKE-A-TRIP-AND-GET-PAID:

```
Unbound-reference to DESTINATIONS in Attributes field (1.00)
Unbound-reference to COST in Attributes field (1.00)
Insufficient-values in Attribute-names field (0.25)
Insufficient-values in Specializations field (1.00)
Insufficient-values in Generalizations field (0.67)
Insufficient-values in Part-of field (1.00)
```

```
Insufficient-values in Parts field (0.75)
Precondition-is-not-satisfied for part GET_REIMBURSED (1.00)
```

When the state-based heuristics are applied to this information, the following expectations result:

```
 Exp82:  Expecting (certainty 0.800):
   MOD:  CREATE the Knac-structure Cost
         Derived from Take_a_trip_and_get_paid and H_S6.

 Exp83:  Expecting (certainty 0.800):
   MOD:  CREATE the Knac-structure Destinations
         Derived from Take_a_trip_and_get_paid and H_S6.

 Exp84:  Expecting (certainty 0.480):
   MOD:  ADD ?New-step<is-an-event-p> to the
             Parts field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_S3.

 Exp85:  Expecting (certainty 0.480):
   MOD:  ADD (Before ?new-step<is-an-event-p> get_reimbursed)
             to the Constraints
             field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_S3.

 Exp146:  Expecting (certainty 0.360):
    MOD:  ADD ?New-part<is-a-knac-structure-p> to the
```

```
        Parts field of Take_a_trip_and_get_paid
    Derived from Take_a_trip_and_get_paid and H_S2.

Exp147: Expecting (certainty 0.480):
   MOD: ADD ?New-part<is-a-knac-structure-p> to the
            Part-of field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S2.

Exp148: Expecting (certainty 0.320):
   MOD: ADD ?New-part<is-a-knac-structure-p> to the
            Generalizations field
            of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S2.

Exp149: Expecting (certainty 0.480):
   MOD: ADD ?New-part<is-a-knac-structure-p> to the
            Specializations field
            of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S2.

Exp150: Expecting (certainty 0.120):
   MOD: ADD ?New-part<is-a-knac-structure-p> to the
            Attribute-names field
            of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_S2.
```

§1.2 *Modification Heuristics*

Although a dialog between a domain expert and a knowledge engineer often contains occasional shifts in context, a certain continuity is generally present. Information provided by the expert is usually related to some other recently discussed information. Therefore, any resulting modifications to the knowledge base may be used to anticipate additional changes.

When an entity is added to the knowledge base, for instance, further description of the entity and of how it fits in with the existing knowledge are often forthcoming. Two heuristics that capture this are:

Heuristic M1: *Detailed information usually follows the introduction of a new entity.*

Heuristic M2: *Context information usually follows the introduction of a new entity.*

For instance, when the event ISSUE-TRAVEL-AUTHORIZATION is added to the knowledge base, the following expectations are generated:

```
Exp271:  Expecting (certainty 0.600):
   MOD:  ADD ?New-value<(lambda (lst)
                           (is-a target-type))> to the
             Generalizations field
             of Issue_travel_authorization
         Derived from Issue_travel_authorization and H_M2.

Exp272:  Expecting (certainty 0.600):
```

```
    MOD:  ADD ?New-value<(lambda (lst)
                             (is-a target-type))> to the
              Part-of field of Issue_travel_authorization
          Derived from Issue_travel_authorization and H_M2.

Exp276:  Expecting (certainty 0.600):
    MOD:  ADD ?New-value<#'(lambda (val)
                              (is-a? val 'event))> to the
              Parts field of Issue_travel_authorization
          Derived from Issue_travel_authorization and H_M1.

Exp277:  Expecting (certainty 0.600):
    MOD:  ADD ?New-value<#'(lambda (val)
                              (is-a? val 'event))> to the
              Specializations field
              of Issue_travel_authorization
          Derived from Issue_travel_authorization and H_M1.

Exp278:  Expecting (certainty 0.600):
    MOD:  ADD ?New-value<is-a-knac-structure-p> to the
               Attributes field of Issue_travel_authorization
          Derived from Issue_travel_authorization and H_M1.
```

Similarly, when information is added to an existing entity, this new information usually gets constrained in some manner.

Heuristic M3: *Attributes are usually constrained after being introduced.*

Heuristic M4: *Parts are usually constrained after being introduced.*

Heuristic M5: *Adding two parts to an entity usually implies a relation between them.*

Adding the step ISSUE-TRAVEL-AUTHORIZATION to the event TAKE-A-TRIP-AND-GET-PAID results in:

```
Exp264:  Expecting (certainty 0.022):
   MOD:  ADD (?Relation<is-a-relationship-p>
              issue_travel_authorization
              ?part2<(lambda (part)
                       (is-a? part 'event))>) to the
             Constraints field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_M4.

Exp265:  Expecting (certainty 0.001):
   MOD:  ADD (?Relation44<is-a-relationship-p> actor
              ?attribute45<is-a-knac-structure-p>) to the
             Constraints field of Take_a_trip_and_get_paid
         Derived from Take_a_trip_and_get_paid and H_M3.

Exp266:  Expecting (certainty 0.600):
   MOD:  ADD (?Relation46<is-a-relationship-p> issue-time
              ?attribute47<is-a-knac-structure-p>) to the
             Constraints field of Issue_travel_authorization
         Derived from Issue_travel_authorization and H_M3.
```

```
Exp267:  Expecting (certainty 0.600):
   MOD:  ADD (?Relation48<is-a-relationship-p> issuee
              ?attribute49<is-a-knac-structure-p>) to the
             Constraints field of Issue_travel_authorization
         Derived from Issue_travel_authorization and H_M3.

Exp268:  Expecting (certainty 0.600):
   MOD:  ADD (?Relation50<is-a-relationship-p> issuer
              ?attribute51<is-a-knac-structure-p>) to the
             Constraints field of Issue_travel_authorization
         Derived from Issue_travel_authorization and H_M3.
```

It is possible to add structure (or even entity) dependent heuristics in order to provide more focused (and therefore more highly rated) expectations. Consider, for instance, two more specific versions of Heuristic M4:

Heuristic M4a: *Parts of Events are usually temporally constrained after being introduced.*

Heuristic M4b: *Parts of Objects are usually spatially constrained after being introduced.*

Thus, the following more specific expectation is also produced:

```
Exp263:  Expecting (certainty 0.036):
   MOD:  ADD (?Temporal-relation<is-a-temporal-relationship-p>
```

```
issue_travel_authorization
?step2<is-an-event-p>) to the
  Constraints field of Take_a_trip_and_get_paid
Derived from Take_a_trip_and_get_paid and H_M4A.
```

Though the removal of an entity from the knowledge base is not a frequent occurrence, any entity containing a reference to the deleted entity may expect a modification to the field that contains this reference.

Heuristic M6: *Pointers to an Entity usually change if the entity is deleted.*

Finally, one last modification heuristic captures the idea of continuity in a very general way:

Heuristic M7: *Recently modified entities may be referenced or modified again.*

The modifications to TAKE-A-TRIP-AND-GET-PAID produce the rather vague expectations:

```
Exp225: Expecting (certainty 0.011):
   MOD: ?ACTION2 Take_a_trip_and_get_paid to/from the
                  ?Field2 field of ?Entity2
        Derived from Take_a_trip_and_get_paid and H_M7.

Exp226: Expecting (certainty 0.011):
   MOD: ?ACTION2 ?Value2 to/from the
                  ?Field2 field of Take_a_trip_and_get_paid
        Derived from Take_a_trip_and_get_paid and H_M7.
```

§*1.3* *Discourse Heuristics*

Research on the analysis of discourses in general, and of knowledge acquisition discourses in particular, provide another potential of means of anticipating modifications [Gro78]. Information provided by the discourse manager concerning the state of the discourse may provide clues about what to expect in the upcoming portions of the dialog. Specifically, information about the topic (and subtopics) of a discourse, changes in context, implicit ordering or relevance of a group of entities, and disambiguation of anaphoric references may all be provided by an appropriately sophisticated discourse manager.

Until $K^{n}{}_{A}c$ is integrated with such a frontend, however, it is not relying on the availability of such information. Therefore, only basic discourse information, such as the topic(s) of the discourse and entity references, is presumed available. Heuristics using this information include:

Heuristic D1: *Entities close to specified topics are likely to be referenced or modified.*

Heuristic D2: *Referenced entities are likely to be modified or referenced again.*

Upon being informed by the discourse manager that the (first) topic of discourse is TRAVEL, expectations of reference or modification to knowledge base entities semantically close to TRAVEL are generated. The certainty of each expectation decreases as the distance from the topic entity increases. These expectations include:

```
Exp21:  Expecting (certainty 0.100):
  MOD:  ?ACTION ?Value to/from the
```

```
                ?Field field of Take_a_trip_and_get_paid
        Derived from Travel and H_D1.

Exp22:  Expecting (certainty 0.100):
  MOD:  ?ACTION Take_a_trip_and_get_paid to/from the
                ?Field field of ?Target<is-a-knac-structure-p>
        Derived from Travel and H_D1.
```

Similarly, when the entity TAKE-A-TRIP is referenced during the discourse, expectations of reference to nearby entities are generated, including:

```
Exp205:  Expecting (certainty 0.245):
   MOD:  ?ACTION ?Value to/from the
                 ?Field field of Take_a_trip_and_get_paid
         Derived from Take_a_trip and H_D2.

Exp206:  Expecting (certainty 0.245):
   MOD:  ?ACTION Take_a_trip_and_get_paid to/from the
                 ?Field field of ?Target
         Derived from Take_a_trip and H_D2.
```

§1.4 *Acquisition Strategy Heuristics*

Computer-based tutoring and automated learning are two areas of research concerned with understanding and describing teaching/learning techniques and strategies. Most knowledge acquisition systems assume a particular strategy and

either force it upon the domain expert (in systems that maintain the initiative) or provide tools tailored to the strategy (in more passive systems).

Different acquisition strategies will result in obtaining different types of information from the user at different times. For instance, an approach in which the acquisition system maintained the initiative and drove the acquisition process in a top-down fashion would produce a very different scenario than a system where the expert presented cases (examples) to the system. The resulting state of the knowledge base may be similar; the modification process would vary.

If the acquisition strategy were known, either by selecting one a priori or by dynamically recognizing it, heuristics particular to that strategy could provide a means of anticipating modifications. At this time, no such heuristics are being employed by $K^{n}{}_{A}c$.

§2. Managing Expectations

As more information is presented by the expert and more modifications to the knowledge base result, $K^{n}{}_{A}c$ is able to generate ever larger numbers of expectations. As the number of expectations grow, the system's ability to focus decreases. Therefore, some method of ranking and pruning the set of expectations is needed.

§*2.1 Rating Expectations*

Each expectation generated by the system has an associated "certainty". This value is assigned when the expectation is generated and may be updated during subsequent time frames. The initial value is a function of the heuristic and the

data from which the expectation arises. Since the certainty of an expectation depends on how it was generated, each heuristic specifies its own function for calculating this value. ([GC86] describes the importance of localizing such functions.) Heuristic D1, for example, which generates candidate matches based on discourse topics, assigns a certainty that is related to the (semantic) distance between the topic and the candidate. Heuristic S2, which uses "holes" in the knowledge to infer the addition of information, relies on the certainty that the information is missing.

In addition to the heuristic's certainty metric, each heuristic has an associated specificity. The more specific a heuristic is, the more certain we may be of the result. Thus, the resulting expectation's certainty is scaled by the heuristic's specificity. This accounts for the greater certainty of an expectation produced by heuristic M4a (which applies to *Events*) than a corresponding modification produced by heuristic M4 (which applies to any type of structure).

Interactions between expectations could also be used to modify these certainties. Mutually consistent expectations, for instance, might increase the certainty of each; conflicting expectations might decrease the certainty. Rather than taking this approach, however, K^nA_c uses this information when evaluating the extent to which a (potential) knowledge base modification is supported by expectations (see Chapter 5 Section §2.2). This permits the inter-expectation support or conflict to be evaluated in the context of a particular modification.

§2.2 *Pruning Expectations*

Once expectations have been assigned a certainty value, the set may be pruned at the end of each time frame, either by thresholding at a specified certainty level

or maintaining a fixed-size set of the highest ranked ones. Two further factors may affect the duration of individual expectations: its effective time frame and its level of satisfaction.

Some expectations are applicable only at the time they are generated; others are always applicable; still others become more or less relevant with the passage of time. To capture this idea, each heuristic assigns a time duration function to the expectations it generates. These functions describe when the expectation is valid and include:

fade: The expectation is most relevant when it is generated and degrades with time. The rate at which it degrades may (optionally) be specified.

increase: The more time that passes without this expectation being satisfied, the more significant it becomes. The rate at which it increases may (optionally) be specified.

until: This expectation is only relevant until some condition (or absolute time) occurs.

while: This expectation is relevant while some condition is true.

after: This expectation only becomes valid after some condition occurs.

for: This expectation remains valid for a prescribed time after its creation.

always: [3] This expectation remains valid forever.

never: [4] This expectation is never valid.

[3]For efficiency, expectations with an *after* time duration function reset the expectation's function to *always* once the specified condition becomes true.

[4]Similarly, *until* and *for* expectations are reset to *never* when the specified condition becomes true.

After each frame, the time duration function of all current expectations are checked. This may either directly result in the removal of some expectations or may modify their certainty and thereby place them below the current threshold.

Expectations may also be removed if they, or their alternative expectations, are satisfied. If an anticipated modification occurs, the expectation of that modification may no longer be needed. However, a choice of whether an expectation persists or expires upon being satisfied is available via the expectation's *persistence* field. (Allowable values are "one-shot" and "endure".)

§3. Frame Segmentation Revisited

In Chapter 3 Section §4., the division of the dialog into time frames was briefly discussed. Without better metrics to guide the division, "convenient" size chunks were selected without much consideration of their content. Choosing time frames containing too little or too much information causes different types of problems.

Consider the addition of the step ISSUE-TRAVEL-AUTHORIZATION to the event TAKE-A-TRIP-AND-GET-PAID. This results in the creation of an expectation of a temporal constraint between the new step and an existing step in the plan. However, since the addition of that temporal constraint occurs in the same discourse frame as the addition of the step, the expectation is not generated in time to support the addition of the constraint.

Reducing the size of the time frames may appear to be the solution. This, however, fails in two ways. First, it will produce an increased number of time frames that contain no useful information (from $K^n{}_A c$'s point of view). More

importantly, the information is not always presented in the "correct" order, either by the expert or as the output of the discourse manager. Furthermore, the optimum order in which the information is integrated is dependent upon the set of heuristics being used. The addition of a step and a temporal constraint on that step may not even be mentioned explicitly in the discourse. Thus, expecting that the information arrive from the discourse manager/parser in the "correct" order is not a reasonable approach.

A fairly simple mechanism is available for overcoming this difficulty. If a match (or set of matches) is assumed to be (potentially) correct, the resulting modifications may be simulated to determine the resulting expectations. These expectations may then be used to re-evaluate the matches under consideration.

It is important to note that, without getting fairly sophisticated, this method will provide only an approximation of truth. For instance, poor matches may tend to lend support to themselves or the results of a match may cause the match itself to be demoted.

An alternate approach, the one currently used by K^nAc, delays the resolution of ambiguous matches until the more certain matches are made. The expectations from the more certain matches are then available to re-evaluate the less clear ones. While this approach misses intra-structure expectations, such as the addition of the step implying the addition of a constraint, it captures many of the expectations between entities within a discourse frame. For example, the recognition of TAKE-A-TRIP as an existing knowledge base description generates expectations that support the match of EVENT-1 to TAKE-A-TRIP-AND-GET-PAID.

§4. What's in a Modification?

In previous sections, modifications to a knowledge base were discussed. In this section, these modifications are more formally defined. Each modification consists of one of four types of action (*create*, *delete*, *add* or *remove*), a target knowledge base entity, a value, and (for *add* and *remove*) a field name. The information contained in these fields is:

create: The target field contains the name of the entity to be created and added to the knowledge base. The value field contains a partial specification of the new entity.

delete: The name of the entity to be deleted from the knowledge base is the only information specified.

add: The name of the entity to be modified, the name of the field and the value to be added to that field are specified.

remove: The name of the entity to be modified, the name of the field and the value to be removed from that field are specified.

The granularity of modifications and expectations was deliberately kept quite small. Each modification describes the creation or deletion of a single entity, or the addition or removal of a single value to or from a field. Each expectation is of a single modification.

Alternatively, each modification could have contained several values, e.g., the addition of several steps to an event. Another scheme could have maintained simple modifications and permitted expectations to include multiple modifications. Either of these schemes would reduce the number of expectations generated,

but increase the complexity of generating, evaluating, maintaining and utilizing them.

The tradeoff is between intra- and inter-expectation processing. Consider two modifications, $M1$ and $M2$. If both were expected by a single expectation (i.e., the conjunct of $M1$ and $M2$ was expected) and only one occurred, the system would require a scheme for representing partial fulfillment (satisfaction) of the expectation. Such problems may be avoided by having $M1$ and $M2$ anticipated by separate expectations. The cost, however, arises when $M1$ and $M2$ are disjuncts instead of conjuncts; the fulfillment of one expectation should must signal the fulfillment of the other. This is accomplished by providing each expectation with a list of alternative expectations; in any of the alternatives are satisfied, so is that expectation.

CHAPTER 7

SYSTEM EVALUATION AND CUSTOMIZATION

The goal of the $K^n{}_Ac$ system is to assimilate information from a domain expert into an existing knowledge base by matching this new information to existing knowledge structures. The existing structures must then be modified to conform to the information provided by the expert. For the discourse fragments on which the system has been tested, appropriate matches were usually selected and the necessary modifications suggested. In the following section, these results are presented and $K^n{}_Ac$'s facilities for examining and evaluating the system's performance are described.

The $K^n{}_Ac$ system is intended to be a testbed for knowledge acquisition and has been designed to permit experimentation with and evaluation of various factors which may contribute to its success. This experimentation includes both customizing portions of the $K^n{}_Ac$ system and modifying the "world" with which it interacts. Specifically, $K^n{}_Ac$ permits modifications to its knowledge acquisition heuristics, to its generation, management and use of expectations, to its matching and match evaluation procedures, and to the knowledge base representing its world model. The latter sections of this chapter describe how each of these aspects of the system may be modified and how the effects of such changes may be observed and measured.

§1. Evaluation of Match Results

When information from the domain expert corresponds to an existing entity in the knowledge base, the goal of the $K^{n}{}_{A}c$ system is to identify the appropriate structure and make the necessary modifications to it. If the expert is describing a entity new to the knowledge base, $K^{n}{}_{A}c$ still attempts to find the "closest" known structure. Because no oracle is available in the current system, $K^{n}{}_{A}c$'s selection of the best match (or matches) cannot be automatically evaluated. Thus, as experimentation with knowledge acquisition heuristics and the pruning of the resulting expectations has proceeded, the resulting matches were always checked (manually) to assure that the system was performing reasonably.

To permit the monitoring of these matches, $K^{n}{}_{A}c$ provides a summary of the matches selected in each discourse frame and, if desired, obtains confirmation of its results from the user. The results for the first five discourse frames are summarized in Figures 20 and 21. (See Appendix C for a more detailed account of the results from these discourse frames.)

In the first discourse frame, no satisfactory matches were found for the object TRAVEL-AUTHORIZATION or for the event ISSUE-TRAVEL-AUTHORIZATION so they were (correctly) presumed to be descriptions of new entities. The event TAKE-A-TRIP was recognized as already being in the knowledge base. There were several close syntactic matches for the specified (but unnamed) event EVENT-1. Expectations generated from the discourse topic of "travel" and from missing information in the candidate entities allowed the "context dependent" match evaluation to correctly select TAKE-A-TRIP-AND-GET-PAID as the best match.

In the second discourse frame, the newly created TRAVEL-AUTHORIZATION was mentioned, reinforcing the system's interest in that portion of the knowl-

```
        Frame 1

TAKE_A_TRIP already in KB
No match found for ISSUE_TRAVEL_AUTHORIZATION
No match found for TRAVEL-AUTHORIZATION
EVENT-1 matches TAKE_A_TRIP_AND_GET_PAID (0.285)
  closest: 0.27 0.30  TAKE_A_TRIP_AND_GET_PAID
           0.27 0.20  VISIT
           0.24 0.17  MAKE_A_RESERVATION
           0.24 0.17  TRAVEL


        Frame 2

TRAVEL-AUTHORIZATION ACCOUNTING already in KB
No match found for SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING


        Frame 3

GRANT FILL_OUT_FORM_FIELD DESTINATION DEPARTMENT-HEAD
      already in KB
No match found for FILL_OUT_TRAVEL_AUTHORIZATION
No match found for SIGN_FORM
  closest: 0.28 0.17  TAKE_A_TRIP_AND_GET_PAID
           0.26 0.22  ISSUE_TRAVEL_AUTHORIZATION
           0.24 0.17  TAKE_A_TRIP
           0.24 0.17  VISIT
           0.22 0.14  TRAVEL
No match found for DEPARTURE-DATE
No match found for RETURN-DATE
No match found for TRUST-FUND
No match found for STATE-FUNDS
No match found for P-I
  closest: 0.42 0.25  TRAVELER
No match found for MEANS-OF-TRANSPORT
No match found for PLANE
No match found for BUS
No match found for PRIVATE-CAR
```

Figure 20: Selected Matches for Discourse Frames 1 through 3

```
        Frame 4

DESTINATION DEPARTURE-DATE MEAL PRIVATE-CAR
FILL_OUT_FORM_FIELD already in KB
No match found for FILL_OUT_TRAVEL_VOUCHER
No match found for TRAVEL-VOUCHER
No match found for COLLECT_RECEIPTS
  closest: 0.24 0.20  SIGN_FORM
           0.24 0.17  VISIT
No match found for TAXI
  closest: 0.31 0.17  PRIVATE-CAR
           0.31 0.16  BUS
           0.31 0.16  PLANE
No match found for RECEIPT-FOR-HOTEL
No match found for RECEIPT-FOR-PLANE


        Frame 5

SECRETARY already in KB
EVENT-2 matches COLLECT_RECEIPTS (0.442)
  closest: 0.49 0.40  COLLECT_RECEIPTS
No match found for EVENT-3
  closest: 0.30 0.25  COLLECT_RECEIPTS
           0.23 0.17  SIGN_FORM
No match found for SUPPLY_TRAVEL_INFORMATION
No match found for GIVE_RECEIPTS_TO_SECRETARY
```

Figure 21: Selected Matches for Discourse Frames 4 through 5

edge base, and the known structure ACCOUNTING was also mentioned, causing the system to focus on that area as well. The new event SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING was added and its references to ACCOUNTING and TRAVEL-AUTHORIZATION further reinforced the systems areas of focus.

Several new entities were added in the third frame, most of which had no close matches in the existing knowledge base. The system did suggest TRAVELER as a possible match for P-I (i.e., a principal investigator) because both are people and because of its focus on entities related to travel. The best syntactic match for the SIGN-FORM event was TAKE-A-TRIP-AND-GET-PAID because of the references in this instance of SIGN-FORM to a "traveler" and a "p.i.". The context dependent matching preferred the more appropriate event ISSUE-TRAVEL-AUTHORIZATION.

In the forth discourse frame, the closest matches to the new object TAXI are PRIVATE-CAR, BUS and PLANE, all introduced in the previous frame. The context dependent rating slightly favors PRIVATE-CAR because it was mentioned again in this discourse frame. The rather strange selection of SIGN-FORM and VISIT as being close to the event COLLECT-RECEIPTS occurs because of the involvement of forms in the first case and the mention of a HOTEL-RECEIPT in the second.

In the final frame shown here, two unnamed events are introduced. The event COLLECT-RECEIPTS was selected as the best match for both of them. However, it was correctly judged (by both the context independent and dependent match ratings) to more closely match EVENT-2. If the binding of EVENT-3, which contains EVENT-2 as its second step, were delayed (as discussed in Chapter 6 Section §3.), the system would attempt to match EVENT-3 to an event containing both TAKE-A-TRIP and COLLECT-RECEIPTS.

§2. Examining the Justification for a Match

In order to understand why K^nAc selected (or did not select) a particular match, the system allows inspection of the match results at various levels of detail. Recall that a match is selected based both on the degree of similarity between a structure described by the domain expert and one already in the knowledge base and on the extent to which the modifications implied by the differences between the two descriptions have been anticipated. Modifications are anticipated through invocations of K^nAc's knowledge acquisition heuristics. The match between the entity EVENT-1[1] described in the discourse and TAKE-A-TRIP-AND-GET-PAID existing in the knowledge base may be explored by displaying information such as that shown in Figures 22 and 23.

To understand the ratings assigned to a structure match (i.e., the "syntactic" context independent rating and the expectation guided context dependent rating), the slot-by-slot matches must be considered. Figure 22 shows such a display for the events EVENT-1 and TAKE-A-TRIP-AND-GET-PAID. For each slot, in addition to the match ratings for that slot, the similarities and differences are also described. The differences imply modifications and the heuristics that have generated expectations which anticipate these modifications are listed.

In order to observe the system's anticipation of these modifications at a finer level of detail, the display shown in Figure 23 lists the modifications required to resolve each of the differences between the structures and the expectations that predicted these modifications. For each expectation, the display contains its certainty, the heuristic which produced it, and the "data" from which it was derived.

[1]The complete descriptions of the entities appear in Figures 10 and 11 in Chapter 3.

```
EVENT-1 vs. TAKE_A_TRIP_AND_GET_PAID

  comparable [independent = 0.271, dependent = 0.299]

  GENERALIZATIONS: [1.000 / 1.000]
  Sets match completely.
  They share: {EVENT}.

  PARTS: [0.036 / 0.146]
  Sets partially match (rating 0.333); 3.6% match likelihood.
  They share: {TAKE_A_TRIP}.
  {ISSUE_TRAVEL_AUTHORIZATION} appears only in first entity.
  {GET_REIMBURSED} appears only in second entity.
  Expectations of implied modifications resulted: {H_D2 H_S2 H_S3}

  ATTRIBUTE-NAMES: [0.002 / 0.039]
  Sets partially match (rating 0.250); 0.2% match likelihood.
  They share: {DESTINATIONS}.
  {ACTOR} appears only in first entity.
  {COST TRAVELER} appear only in second entity.
  Expectations of implied modifications resulted: {H_D2}

  CONSTRAINTS: [0.048 / 0.009]
  Constraints were CONSISTENT.
  Match rating 0.048;
  They share: (DESTINATIONS = TAKE_A_TRIP.DESTINATIONS)
  Minimum additions to the first are:
    (TAKE_A_TRIP BEFORE GET_REIMBURSED)
    (COST = GET_REIMBURSED.AMOUNT)
    (GET_REIMBURSED.AMOUNT = TAKE_A_TRIP.COST)
    (GET_REIMBURSED.RECIPIENT = TRAVELER)
    (TRAVELER = ACTOR)
  Minimum additions to the second are:
    (DESTINATION OUTSIDE STATE)
    (ISSUE_TRAVEL_AUTHORIZATION BEFORE TAKE_A_TRIP)
    (ACTOR = ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
    (ISSUE_TRAVEL_AUTHORIZATION.ISSUEE = GET_REIMBURSED.RECIPIENT)
  Expectations of implied modifications resulted: {H_D2 H_S3}
```

Figure 22: Justification for a Structure Match in Frame 1

```
PARTS: [0.036 / 0.146]
Sets partially match (rating 0.333); 3.6% match likelihood.
They share: {TAKE_A_TRIP}.
{ISSUE_TRAVEL_AUTHORIZATION} appears only in first entity.
  Implied modifications:
  MOD1156: ADD Issue_travel_authorization to the Parts field of
           Take_a_trip_and_get_paid  (certainty: 0.036)
      Expected: 0.332(average) 0.480(maximum) 0.719(independent)

      Exp19:  Expecting (certainty 0.157):
       MOD: ?ACTION ?Value to/from the ?Field field of
                     Take_a_trip_and_get_paid
       Derived from H_D2 and TAKE_A_TRIP already in KB (1.000).

      Exp114:  Expecting (certainty 0.480):
       MOD: ADD ?New-step<is-an-event-p> to the Parts field of
                Take_a_trip_and_get_paid
       Derived from H_S3 and
                STATE: PRECONDITION-IS-NOT-SATISFIED for
                       entity TAKE_A_TRIP_AND_GET_PAID
                       Field: PARTS   Values: (GET_REIMBURSED).

      Exp169:  Expecting (certainty 0.360):
       MOD: ADD ?New-part<is-a-knac-structure-p> to the Parts
                field of Take_a_trip_and_get_paid
       Derived from H_S2 and
                STATE: INSUFFICIENT-VALUES (certainty 0.750) for
                       entity TAKE_A_TRIP_AND_GET_PAID.
                       Field: PARTS   Values: 3/4.

{GET_REIMBURSED} appears only in second entity.
  Implied modifications:
  MOD1157: REMOVE Get_reimbursed from the Parts field of
                  Take_a_trip_and_get_paid  (certainty: 0.036)
      Expected: 0.157(average) 0.157(maximum) 0.157(independent)

      Exp19:  Expecting (certainty 0.157):
      MOD: ?ACTION ?Value to/from the ?Field field of
                    Take_a_trip_and_get_paid
      Derived from H_D2 and TAKE_A_TRIP already in KB (1.000).
```

Figure 23: Support for the "Parts" Field Match

§3. Knowledge Acquisition Heuristics

Since K^nAc's knowledge acquisition heuristics serve to anticipate modifications to an existing knowledge base, their performance may be measured in terms of the effectiveness of the expectations they produce. Because there may be interactions within a set of heuristics, the effectiveness of both the individual heuristics and of the entire set should be considered.

The performance of each heuristic may be evaluated in terms of how often it is invoked, how many of these invocations successfully produce expectations and how many expectations are produced. A "local" and a more "global" measure of the quality of the resulting expectations may be obtained from the system's "certainty" in the expectations and the extent to which they are (eventually) fulfilled. Specifically, the factors considered in evaluating a heuristic's performance are:

- *Context specificity:* In order for a heuristic to produce expectations, its conditions must be satisfied. Heuristics with more specific conditions will be successfully invoked less often. The *context specificity* of a heuristic is the percentage of invocations in which the heuristic's conditions are satisfied. A heuristic that is too specific may never get successfully invoked; one that is too general may flood the system with poor quality expectations.

- *Number of expectations per invocation:*[2] A successfully invoked heuristic can produce one or more expectations. Since controlling the explosion of expectations is a major task in the K^nAc system, those heuristics that

[2]Because of the way in which the satisfaction of *conjunctive* constraints is determined, the number of "invocations" of those heuristics whose *condition* contains such constraints is not counted correctly. Currently, this affects heuristics H_5, H_5A, and H_5B.

produce a large number of expectations, or those whose production goes up as the discourse progresses, may need to be identified.

- *Certainty of resulting expectations:* The "certainty" of the expectations produced by a heuristic depend on the quality of the data on which it is invoked and on the specificity of the heuristic (as described in Chapter 6 Section §2.). The average certainty of the resulting expectations, while not necessarily a valid measure of their eventual usefulness (see "Accuracy" below), is still an important measure. A heuristic that consistently produces expectations of low certainty may be getting invoked only on data in which the system has little confidence. Further, these low certainty expectations will tend to be overpowered by higher rated ones, thereby decreasing the overall contribution of the heuristic.

- *Accuracy of resulting expectations:* Since expectations attempt to anticipate subsequent modifications to the knowledge base, an expectation is "fulfilled" if the anticipated modification actually occurs. The *accuracy* of a heuristic is the percentage of expectations it has produced that have, at a given point in time, been fulfilled. Since, at any point, some expectations that will eventually be fulfilled have not yet been, this measures understates the heuristic's true accuracy.

The above measurements of the heuristics' performance may be aggregated over time and over classes of heuristics. By plotting the above values against time (measured in discourse frames), changes in a heuristic's performance as the discourse progresses may be observed. Displaying the performance of classes of heuristics over time may reveal patterns that are not as apparent in the performance of individual heuristics. The most obvious groupings are by heuristic type

	Times called	Times ok	Success rate	Avg. no. of exps	Avg. exp certainty	Accuracy of exps
H_D1:	67	1	0.01	21.00	0.17	0.00
H_D2:	67	66	0.99	14.64	0.09	0.03
H_M1:	164	24	0.15	3.00	0.31	0.01
H_M2:	164	24	0.15	2.00	0.31	0.02
H_M3:	164	25	0.15	1.00	0.32	0.00
H_M4:	164	17	0.10	1.00	0.28	0.00
H_M4A:	164	15	0.09	1.00	0.43	0.00
H_M4B:	164	2	0.01	1.00	0.64	0.00
H_M5:	5	0	0.00	–	–	–
H_M5A:	5	0	0.00	–	–	–
H_M5B:	5	0	0.00	–	–	–
H_M6:	164	0	0.00	–	–	–
H_M7:	164	164	1.00	2.00	0.15	0.05
H_S2:	711	469	0.66	1.00	0.21	0.01
H_S3:	711	3	0.00	2.00	0.13	0.17
H_S4:	711	0	0.00	–	–	–
H_S6:	711	179	0.25	1.00	0.35	0.00

Figure 24: Heuristic Statistics through Discourse Frame 5

(e.g., discourse, state, modification), but other groupings may be produced as desired.

The statistics of heuristic performance through discourse frame 5 (shown in Figure 24) begin to reveal different characteristics among the heuristics being used. For instance, heuristics D2 (*"Referenced entities are likely to be modified or referenced again."*), M7 (*"Recently modified entities may be modified again or referenced."*) and S2 (*"Field with too few components will be augmented."*), being rather broad in scope, are responsible for most of the expectations produced. Heuristics such as M4A (*"Parts of Events are usually temporally constrained after being introduced."*) are somewhat more focused, producing fewer expectations, but with higher certainties. Because many correctly anticipated modifications

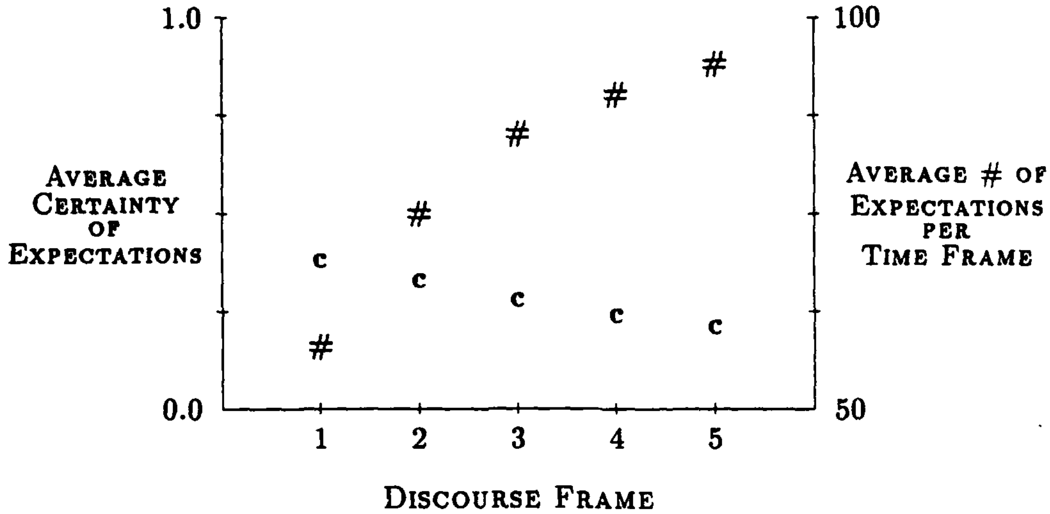


Figure 25: Number and Certainty of Expectations from Heuristic S2

have not yet been made to the knowledge base, the measured accuracy of all of the heuristics is still very low.

The performance of a heuristic can change as the discourse progresses. This can be seen for heuristic S2 (described above) in Figure 25. As more entities in the knowledge base are brought into focus (i.e., are considered as potential matches for the discourse entities), incompleteness and inconsistencies in these entities are recognized. Those structure slots suspected of containing too few values caus' heuristic S2 to generate expectations of values being added to those slots. However, as the set of potential match candidates grows, the system's confidence in each of these entities decreases. Because the certainties of the expectations produced by heuristic S2 depend on the ratings of these entities, the certainties of the resulting expectations also decrease. Figure 25 shows this increase in the heuristic's productivity and the corresponding drop in the average certainty of its expectations.

In addition to considering the performance of individual heuristics and classes of heuristics, the effectiveness of an entire set of heuristics may also be considered. In order to reveal interactions between heuristics, the assimilation of the discourse may be performed with one or more heuristics deactivated. If similar expectations result, the removed heuristic(s) may have been contributing little or providing only redundant information. If certain expectations are missed and the removed heuristic was not responsible for these expectations in the initial assimilation, an implicit dependency among the heuristics may be inferred. Such ablation experiments have not yet been performed.

§4. Expectations

As described in Chapter 6 Section §2., controlling the growth of the number of "current" expectations (i.e., the set of expectations considered relevant at a given time) is an important part of the $K^n{}_Ac$ system. If too many expectations are allowed, the system's ability to discriminate among potential structure matches diminishes, since most of the implied modifications will be supported by some expectation(s). Furthermore, the system's performance is degraded, both in terms of results and speed of execution. If too few expectations are permitted, the "correct" matches may missed because necessary modifications were not anticipated.

In order to evaluate the system's management of expectations, $K^n{}_Ac$ permits the display of the number of expectations created in each time frame (both the complete set and those selected as "curr nt"), the cumulative totals up to a given discourse frame, and the average certainties of the expectations for each frame. (Figure 26 shows these statistics for discourse frames 1 through 5.) Expectations

	Threshold	Total	Current	Quality	Accuracy
Frame 1	0.15	262	112	0.48	0.03
Frame 2	0.15	472	294	0.40	0.03
Frame 3	0.15	1378	589	0.32	0.02
Frame 4	0.15	1907	927	0.27	0.03
Frame 5	0.15	2148	732	0.29	0.01

Figure 26: Expectation Statistics (Frames 1 through 5)

may be viewed individually or may be grouped based on attributes such as the type of modification involved, the heuristic causing it, the data on which it was invoked, or the target knowledge structure or field.

Controlling the number of "current" expectations, using the expectation certainty threshold and the (default) rate of decrease of expectation certainty (for those expectations whose certainty fades with time) is still more of an art than a science. While the system has been under development, these parameters have been set so that number of expectations considered has been kept fairly high. The rationale for this has been that it is easier to understand the system's performance if excess "data" were available rather than trying to explain "missed" matches based on expectations that were not considered. The cost of this decision has been very slow system performance. More recent experiments have been conducted with more severe expectation thresholds.

Figure 27 shows the rate of growth in the total number of expectations produced by the system and in those selected as "current" for the first five discourse frames. The expectation certainty threshold was set at 0.15 and the rate at which certainties were degraded each frame (for those expectations designated to "fade") was 0.2.

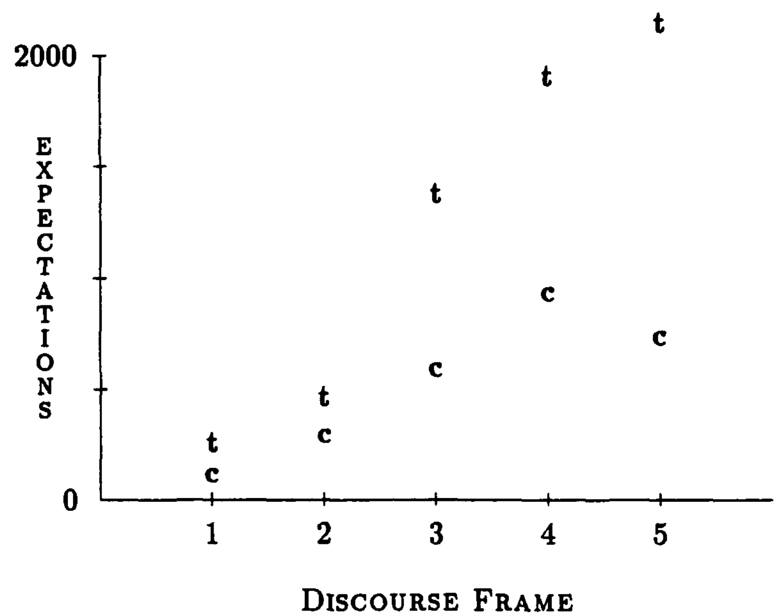


Figure 27: Total and "Current" Expectations

While *expectation management* successfully prunes away a large number of lower rated expectations, the number remaining is still quite high. This large number of expectations is one of the major factors contributing to extremely slow system performance. Raising the expectation certainty threshold or, to a somewhat lesser extent, increasing the rate of "fade" causes the system to fail to anticipate certain *modifications* crucial to recognizing some of the "appropriate" matches in the sample discourse. Therefore, in order for the overall system performance to be significantly improved, either fewer expectations must be produced or the selection of "correct" expectations must become more sophisticated.

§1. Evaluation of Match Results

When information from the domain expert corresponds to an existing entity in the knowledge base, the goal of the $K^n{}_A c$ system is to identify the appropriate structure and make the necessary modifications to it. If the expert is describing a entity new to the knowledge base, $K^n{}_A c$ still attempts to find the "closest" known structure. Because no oracle is available in the current system, $K^n{}_A c$'s selection of the best match (or matches) cannot be automatically evaluated. Thus, as experimentation with knowledge acquisition heuristics and the pruning of the resulting expectations has proceeded, the resulting matches were always checked (manually) to assure that the system was performing reasonably.

To permit the monitoring of these matches, $K^n{}_A c$ provides a summary of the matches selected in each discourse frame and, if desired, obtains confirmation of its results from the user. The results for the first five discourse frames are summarized in Figures 20 and 21. (See Appendix C for a more detailed account of the results from these discourse frames.)

In the first discourse frame, no satisfactory matches were found for the object TRAVEL-AUTHORIZATION or for the event ISSUE-TRAVEL-AUTHORIZATION so they were (correctly) presumed to be descriptions of new entities. The event TAKE-A-TRIP was recognized as already being in the knowledge base. There were several close syntactic matches for the specified (but unnamed) event EVENT-1. Expectations generated from the discourse topic of "travel" and from missing information in the candidate entities allowed the "context dependent" match evaluation to correctly select TAKE-A-TRIP AND GET PAID as the best match.

In the second discourse frame, the newly created TRAVEL-AUTHORIZATION was mentioned, reinforcing the system's interest in that portion of the knowl-

```
        Frame 1

TAKE_A_TRIP already in KB
No match found for ISSUE_TRAVEL_AUTHORIZATION
No match found for TRAVEL-AUTHORIZATION
EVENT-1 matches TAKE_A_TRIP_AND_GET_PAID (0.285)
  closest: 0.27 0.30  TAKE_A_TRIP_AND_GET_PAID
           0.27 0.20  VISIT
           0.24 0.17  MAKE_A_RESERVATION
           0.24 0.17  TRAVEL


        Frame 2

TRAVEL-AUTHORIZATION ACCOUNTING already in KB
No match found for SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING


        Frame 3

GRANT FILL_OUT_FORM_FIELD DESTINATION DEPARTMENT-HEAD
      already in KB
No match found for FILL_OUT_TRAVEL_AUTHORIZATION
No match found for SIGN_FORM
  closest: 0.28 0.17  TAKE_A_TRIP_AND_GET_PAID
           0.26 0.22  ISSUE_TRAVEL_AUTHORIZATION
           0.24 0.17  TAKE_A_TRIP
           0.24 0.17  VISIT
           0.22 0.14  TRAVEL
No match found for DEPARTURE-DATE
No match found for RETURN-DATE
No match found for TRUST-FUND
No match found for STATE-FUNDS
No match found for P-I
  closest: 0.42 0.25  TRAVELER
No match found for MEANS-OF-TRANSPORT
No match found for PLANE
No match found for BUS
No match found for PRIVATE-CAR
```

Figure 20: Selected Matches for Discourse Frames 1 through 3

```
Frame 4

DESTINATION DEPARTURE-DATE MEAL PRIVATE-CAR
FILL_OUT_FORM_FIELD already in KB
No match found for FILL_OUT_TRAVEL_VOUCHER
No match found for TRAVEL-VOUCHER
No match found for COLLECT_RECEIPTS
  closest: 0.24 0.20  SIGN_FORM
           0.24 0.17  VISIT
No match found for TAXI
  closest: 0.31 0.17  PRIVATE-CAR
           0.31 0.16  BUS
           0.31 0.16  PLANE
No match found for RECEIPT-FOR-HOTEL
No match found for RECEIPT-FOR-PLANE


Frame 5

SECRETARY already in KB
EVENT-2 matches COLLECT_RECEIPTS (0.442)
  closest: 0.49 0.40  COLLECT_RECEIPTS
No match found for EVENT-3
  closest: 0.30 0.25  COLLECT_RECEIPTS
           0.23 0.17  SIGN_FORM
No match found for SUPPLY_TRAVEL_INFORMATION
No match found for GIVE_RECEIPTS_TO_SECRETARY
```

Figure 21: Selected Matches for Discourse Frames 4 through 5

edge base, and the known structure ACCOUNTING was also mentioned, causing the system to focus on that area as well. The new event SEND-TRAVEL-AUTHORIZATION-TO-ACCOUNTING was added and its references to ACCOUNTING and TRAVEL-AUTHORIZATION further reinforced the systems areas of focus.

Several new entities were added in the third frame, most of which had no close matches in the existing knowledge base. The system did suggest TRAVELER as a possible match for P-I (i.e., a principal investigator) because both are people and because of its focus on entities related to travel. The best syntactic match for the SIGN-FORM event was TAKE-A-TRIP-AND-GET-PAID because of the references in this instance of SIGN-FORM to a "traveler" and a "p.i.". The context dependent matching preferred the more appropriate event ISSUE-TRAVEL-AUTHORIZATION.

In the forth discourse frame, the closest matches to the new object TAXI are PRIVATE-CAR, BUS and PLANE, all introduced in the previous frame. The context dependent rating slightly favors PRIVATE-CAR because it was mentioned again in this discourse frame. The rather strange selection of SIGN-FORM and VISIT as being close to the event COLLECT-RECEIPTS occurs because of the involvement of forms in the first case and the mention of a HOTEL-RECEIPT in the second.

In the final frame shown here, two unnamed events are introduced. The event COLLECT-RECEIPTS was selected as the best match for both of them. However, it was correctly judged (by both the context independent and dependent match ratings) to more closely match EVENT-2. If the binding of EVENT-3, which contains EVENT-2 as its second step, were delayed (as discussed in Chapter 6 Section §3.), the system would attempt to match EVENT-3 to an event containing both TAKE-A-TRIP and COLLECT-RECEIPTS.

§2. Examining the Justification for a Match

In order to understand why K^n_Ac selected (or did not select) a particular match, the system allows inspection of the match results at various levels of detail. Recall that a match is selected based both on the degree of similarity between a structure described by the domain expert and one already in the knowledge base and on the extent to which the modifications implied by the differences between the two descriptions have been anticipated. Modifications are anticipated through invocations of K^n_Ac's knowledge acquisition heuristics. The match between the entity EVENT-1[1] described in the discourse and TAKE-A-TRIP-AND-GET-PAID existing in the knowledge base may be explored by displaying information such as that shown in Figures 22 and 23.

To understand the ratings assigned to a structure match (i.e., the "syntactic" context independent rating and the expectation guided context dependent rating), the slot-by-slot matches must be considered. Figure 22 shows such a display for the events EVENT-1 and TAKE-A-TRIP-AND-GET-PAID. For each slot, in addition to the match ratings for that slot, the similarities and differences are also described. The differences imply modifications and the heuristics that have generated expectations which anticipate these modifications are listed.

In order to observe the system's anticipation of these modifications at a finer level of detail, the display shown in Figure 23 lists the modifications required to resolve each of the differences between the structures and the expectations that predicted these modifications. For each expectation, the display contains its certainty, the heuristic which produced it, and the "data" from which it was derived.

[1]The complete descriptions of the entities appear in Figures 10 and 11 in Chapter 3.

```
EVENT-1 vs. TAKE_A_TRIP_AND_GET_PAID

  comparable [independent = 0.271, dependent = 0.299]

  GENERALIZATIONS: [1.000 / 1.000]
  Sets match completely.
  They share: {EVENT}.

  PARTS: [0.036 / 0.146]
  Sets partially match (rating 0.333); 3.6% match likelihood.
  They share: {TAKE_A_TRIP}.
  {ISSUE_TRAVEL_AUTHORIZATION} appears only in first entity.
  {GET_REIMBURSED} appears only in second entity.
  Expectations of implied modifications resulted: {H_D2 H_S2 H_S3}

  ATTRIBUTE-NAMES: [0.002 / 0.039]
  Sets partially match (rating 0.250); 0.2% match likelihood.
  They share: {DESTINATIONS}.
  {ACTOR} appears only in first entity.
  {COST TRAVELER} appear only in second entity.
  Expectations of implied modifications resulted: {H_D2}

  CONSTRAINTS: [0.048 / 0.009]
  Constraints were CONSISTENT.
  Match rating 0.048;
  They share: (DESTINATIONS = TAKE_A_TRIP.DESTINATIONS)
  Minimum additions to the first are:
    (TAKE_A_TRIP BEFORE GET_REIMBURSED)
    (COST = GET_REIMBURSED.AMOUNT)
    (GET_REIMBURSED.AMOUNT = TAKE_A_TRIP.COST)
    (GET_REIMBURSED.RECIPIENT = TRAVELER)
    (TRAVELER = ACTOR)
  Minimum additions to the second are:
    (DESTINATION OUTSIDE STATE)
    (ISSUE_TRAVEL_AUTHORIZATION BEFORE TAKE_A_TRIP)
    (ACTOR = ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
    (ISSUE_TRAVEL_AUTHORIZATION.ISSUEE = GET_REIMBURSED.RECIPIENT)
  Expectations of implied modifications resulted: {H_D2 H_S3}
```

Figure 22: Justification for a Structure Match in Frame 1

```
PARTS: [0.036 / 0.146]
Sets partially match (rating 0.333); 3.6% match likelihood.
They share: {TAKE_A_TRIP}.
{ISSUE_TRAVEL_AUTHORIZATION} appears only in first entity.
  Implied modifications:
  MOD1156: ADD Issue_travel_authorization to the Parts field of
           Take_a_trip_and_get_paid  (certainty: 0.036)
     Expected: 0.332(average) 0.480(maximum) 0.719(independent)

     Exp19:  Expecting (certainty 0.157):
      MOD: ?ACTION ?Value to/from the ?Field field of
                   Take_a_trip_and_get_paid
      Derived from H_D2 and TAKE_A_TRIP already in KB (1.000).

     Exp114:  Expecting (certainty 0.480):
      MOD: ADD ?New-step<is-an-event-p> to the Parts field of
               Take_a_trip_and_get_paid
      Derived from H_S3 and
              STATE: PRECONDITION-IS-NOT-SATISFIED for
                     entity TAKE_A_TRIP_AND_GET_PAID
                     Field: PARTS   Values: (GET_REIMBURSED).

     Exp169:  Expecting (certainty 0.360):
      MOD: ADD ?New-part<is-a-knac-structure-p> to the Parts
               field of Take_a_trip_and_get_paid
      Derived from H_S2 and
              STATE: INSUFFICIENT-VALUES (certainty 0.750) for
                     entity TAKE_A_TRIP_AND_GET_PAID.
                     Field: PARTS   Values: 3/4.

{GET_REIMBURSED} appears only in second entity.
  Implied modifications:
  MOD1157: REMOVE Get_reimbursed from the Parts field of
                  Take_a_trip_and_get_paid  (certainty: 0.036)
      Expected: 0.157(average) 0.157(maximum) 0.157(independent)

      Exp19:  Expecting (certainty 0.157):
      MOD: ?ACTION ?Value to/from the ?Field field of
                   Take_a_trip_and_get_paid
      Derived from H_D2 and TAKE_A_TRIP already in KB (1.000).
```

Figure 23: Support for the "Parts" Field Match

§3. Knowledge Acquisition Heuristics

Since $K^n{}_Ac$'s knowledge acquisition heuristics serve to anticipate modifications to an existing knowledge base, their performance may be measured in terms of the effectiveness of the expectations they produce. Because there may be interactions within a set of heuristics, the effectiveness of both the individual heuristics and of the entire set should be considered.

The performance of each heuristic may be evaluated in terms of how often it is invoked, how many of these invocations successfully produce expectations and how many expectations are produced. A "local" and a more "global" measure of the quality of the resulting expectations may be obtained from the system's "certainty" in the expectations and the extent to which they are (eventually) fulfilled. Specifically, the factors considered in evaluating a heuristic's performance are:

- *Context specificity:* In order for a heuristic to produce expectations, its conditions must be satisfied. Heuristics with more specific conditions will be successfully invoked less often. The *context specificity* of a heuristic is the percentage of invocations in which the heuristic's conditions are satisfied. A heuristic that is too specific may never get successfully invoked; one that is too general may flood the system with poor quality expectations.

- *Number of expectations per invocation:*[2] A successfully invoked heuristic can produce one or more expectations. Since controlling the explosion of expectations is a major task in the $K^n{}_Ac$ system, those heuristics that

[2]Because of the way in which the satisfaction of *conjunctive* constraints is determined, the number of "invocations" of those heuristics whose *condition* contains such constraints is not counted correctly. Currently, this affects heuristics H 5, H 5A, and H 5B.

produce a large number of expectations, or those whose production goes up as the discourse progresses, may need to be identified.

- *Certainty of resulting expectations:* The "certainty" of the expectations produced by a heuristic depend on the quality of the data on which it is invoked and on the specificity of the heuristic (as described in Chapter 6 Section §2.). The average certainty of the resulting expectations, while not necessarily a valid measure of their eventual usefulness (see "Accuracy" below), is still an important measure. A heuristic that consistently produces expectations of low certainty may be getting invoked only on data in which the system has little confidence. Further, these low certainty expectations will tend to be overpowered by higher rated ones, thereby decreasing the overall contribution of the heuristic.

- *Accuracy of resulting expectations:* Since expectations attempt to anticipate subsequent modifications to the knowledge base, an expectation is "fulfilled" if the anticipated modification actually occurs. The *accuracy* of a heuristic is the percentage of expectations it has produced that have, at a given point in time, been fulfilled. Since, at any point, some expectations that will eventually be fulfilled have not yet been, this measures understates the heuristic's true accuracy.

The above measurements of the heuristics' performance may be aggregated over time and over classes of heuristics. By plotting the above values against time (measured in discourse frames), changes in a heuristic's performance as the discourse progresses may be observed. Displaying the performance of classes of heuristics over time may reveal patterns that are not as apparent in the performance of individual heuristics. The most obvious groupings are by heuristic type

	Times called	Times ok	Success rate	Avg. no. of exps	Avg. exp certainty	Accuracy of exps
H_D1:	67	1	0.01	21.00	0.17	0.00
H_D2:	67	66	0.99	14.64	0.09	0.03
H_M1:	164	24	0.15	3.00	0.31	0.01
H_M2:	164	24	0.15	2.00	0.31	0.02
H_M3:	164	25	0.15	1.00	0.32	0.00
H_M4:	164	17	0.10	1.00	0.28	0.00
H_M4A:	164	15	0.09	1.00	0.43	0.00
H_M4B:	164	2	0.01	1.00	0.64	0.00
H_M5:	5	0	0.00	–	–	–
H_M5A:	5	0	0.00	–	–	–
H_M5B:	5	0	0.00	–	–	–
H_M6:	164	0	0.00	–	–	–
H_M7:	164	164	1.00	2.00	0.15	0.05
H_S2:	711	469	0.66	1.00	0.21	0.01
H_S3:	711	3	0.00	2.00	0.13	0.17
H_S4:	711	0	0.00	–	–	–
H_S6:	711	179	0.25	1.00	0.35	0.00

Figure 24: Heuristic Statistics through Discourse Frame 5

(e.g., discourse, state, modification), but other groupings may be produced as desired.

The statistics of heuristic performance through discourse frame 5 (shown in Figure 24) begin to reveal different characteristics among the heuristics being used. For instance, heuristics D2 (*"Referenced entities are likely to be modified or referenced again."*), M7 (*"Recently modified entities may be modified again or referenced."*) and S2 (*"Field with too few components will be augmented."*), being rather broad in scope, are responsible for most of the expectations produced. Heuristics such as M4A (*"Parts of Events are usually temporally constrained after being introduced."*) are somewhat more focused, producing fewer expectations, but with higher certainties. Because many correctly anticipated modifications

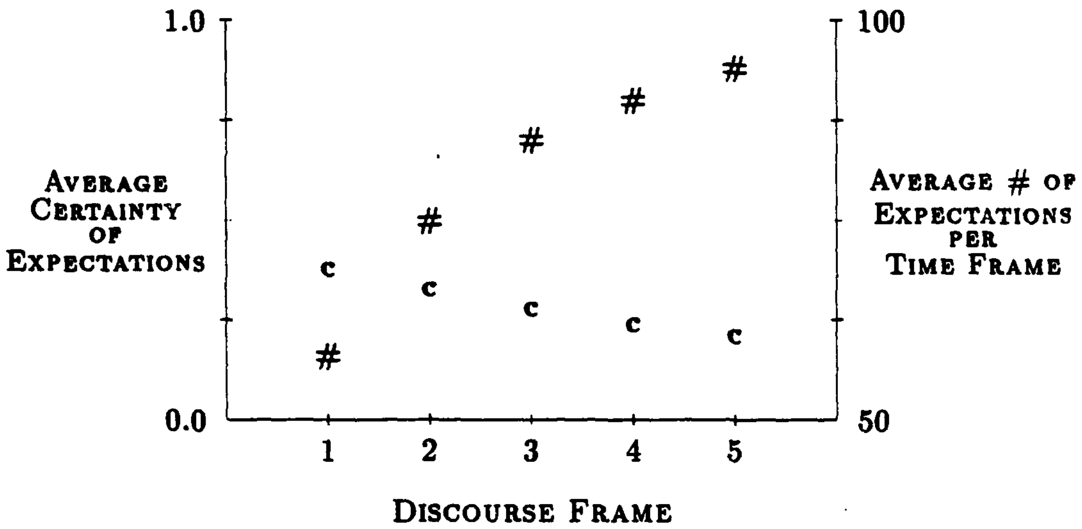


Figure 25: Number and Certainty of Expectations from Heuristic S2

have not yet been made to the knowledge base, the measured accuracy of all of the heuristics is still very low.

The performance of a heuristic can change as the discourse progresses. This can be seen for heuristic S2 (described above) in Figure 25. As more entities in the knowledge base are brought into focus (i.e., are considered as potential matches for the discourse entities), incompleteness and inconsistencies in these entities are recognized. Those structure slots suspected of containing too few values cause heuristic S2 to generate expectations of values being added to those slots. However, as the set of potential match candidates grows, the system's confidence in each of these entities decreases. Because the certainties of the expectations produced by heuristic S2 depend on the ratings of these entities, the certainties of the resulting expectations also decrease. Figure 25 shows this increase in the heuristic's productivity and the corresponding drop in the average certainty of its expectations.

In addition to considering the performance of individual heuristics and classes of heuristics, the effectiveness of an entire set of heuristics may also be considered. In order to reveal interactions between heuristics, the assimilation of the discourse may be performed with one or more heuristics deactivated. If similar expectations result, the removed heuristic(s) may have been contributing little or providing only redundant information. If certain expectations are missed and the removed heuristic was not responsible for these expectations in the initial assimilation, an implicit dependency among the heuristics may be inferred. Such ablation experiments have not yet been performed.

§4. Expectations

As described in Chapter 6 Section §2., controlling the growth of the number of "current" expectations (i.e., the set of expectations considered relevant at a given time) is an important part of the K$^{n}_{A}$c system. If too many expectations are allowed, the system's ability to discriminate among potential structure matches diminishes, since most of the implied modifications will be supported by some expectation(s). Furthermore, the system's performance is degraded, both in terms of results and speed of execution. If too few expectations are permitted, the "correct" matches may missed because necessary modifications were not anticipated.

In order to evaluate the system's management of expectations, K$^{n}_{A}$c permits the display of the number of expectations created in each time frame (both the complete set and those selected as "current"), the cumulative totals up to a given discourse frame, and the average certainties of the expectations for each frame. (Figure 26 shows these statistics for discourse frames 1 through 5.) Expectations

	Threshold	Total	Current	Quality	Accuracy
Frame 1	0.15	262	112	0.48	0.03
Frame 2	0.15	472	294	0.40	0.03
Frame 3	0.15	1378	589	0.32	0.02
Frame 4	0.15	1907	927	0.27	0.03
Frame 5	0.15	2148	732	0.29	0.01

Figure 26: Expectation Statistics (Frames 1 through 5)

may be viewed individually or may be grouped based on attributes such as the type of modification involved, the heuristic causing it, the data on which it was invoked, or the target knowledge structure or field.

Controlling the number of "current" expectations, using the expectation certainty threshold and the (default) rate of decrease of expectation certainty (for those expectations whose certainty fades with time) is still more of an art than a science. While the system has been under development, these parameters have been set so that number of expectations considered has been kept fairly high. The rationale for this has been that it is easier to understand the system's performance if excess "data" were available rather than trying to explain "missed" matches based on expectations that were not considered. The cost of this decision has been very slow system performance. More recent experiments have been conducted with more severe expectation thresholds.

Figure 27 shows the rate of growth in the total number of expectations produced by the system and in those selected as "current" for the first five discourse frames. The expectation certainty threshold was set at 0.15 and the rate at which certainties were degraded each frame (for those expectations designated to "fade") was 0.2.

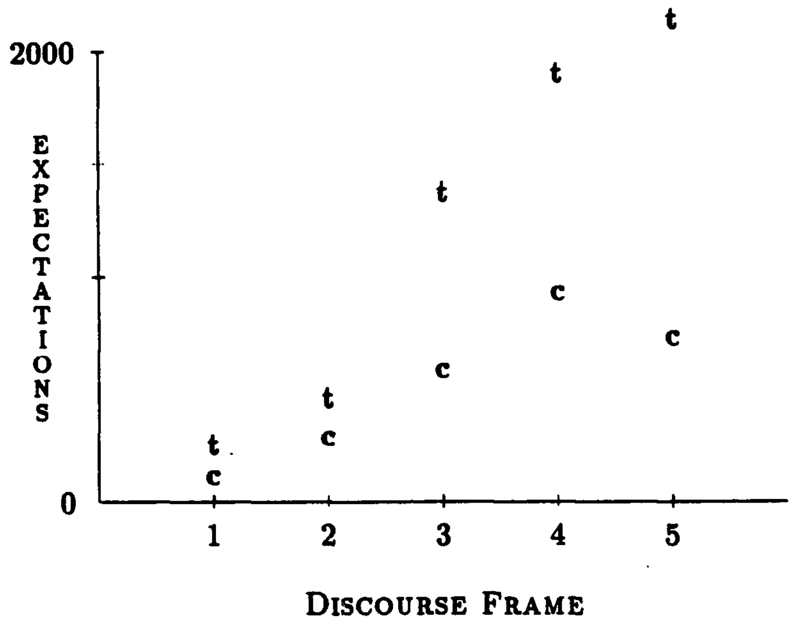


Figure 27: Total and "Current" Expectations

While *expectation management* successfully prunes away a large number of lower rated expectations, the number remaining is still quite high. This large number of expectations is one of the major factors contributing to extremely slow system performance. Raising the expectation certainty threshold or, to a somewhat lesser extent, increasing the rate of "fade" causes the system to fail to anticipate certain *modifications* crucial to recognizing some of the "appropriate" matches in the sample discourse. Therefore, in order for the overall system performance to be significantly improved, either fewer expectations must be produced or the selection of "correct" expectations must become more sophisticated.

§5. Rating of Structure Matches

In addition to the effects of expectations and the heuristics which produce them, the selection of a best match is also affected by the match evaluation process. The way in which $K^{n}{}_{A}c$ combines its slot-by-slot match results to determine an overall structure match value can be modified to better reflect the (semantic) importance of particular structure slots. If, in a particular representation and application domain, strong matches in certain slots prove to be a better indicator of a correct structure match than do other slots, the structure match evaluation function can be modified so as to place more emphasis on those slots. For instance, a match in the *generalizations* slot (implying that the two structures belong to the same class) may prove more significant than a match in the *constraints* slot. Experimenting with these weights permits the significance of a match in each slot of the knowledge structure to be evaluated.

§6. Content and Format of the Knowledge Base

The acquisition process is also affected by both the content of the knowledge base and the representation used to encode it. $K^{n}{}_{A}c$ was designed to be applicable to a variety of frame-based representations. This was accomplished, in part, by isolating the acquisition process from the semantics of the knowledge description frames. For instance, though the structure matching and match evaluation, the inheritance of field values, the determination of semantic "closeness" within the knowledge base, and certain acquisition heuristics are all dependent on particular structures with particular fields, the field-specific aspects of these components were parameterized to simplify the transition to another representation and to

permit experimentation within the current one.

The "completeness" of the knowledge base can also affect the character of the knowledge assimilation process. If little domain knowledge is available, such as early in the construction of a knowledge base, or little *relevant* information is known, as when extending a knowledge base to a new domain, $K^{n}{}_{A}c$ is less able to locate structures into which the new domain information can be assimilated. On the other hand, as the knowledge base becomes more complete, the problem shifts to one of being able to choose from among the many potential matches. By using the context mechanism described in Chapter 4 Section §4. to exclude portions of the existing knowledge, $K^{n}{}_{A}c$ allows experimentation with knowledge bases at varying stages of completion.

Chapter 8

Conclusions

The work presented here has attempted to address one aspect of the knowledge acquisition process. In this final chapter, the goals of this work are reviewed, the successes and the shortcomings are examined, and the still unanswered questions are explored.

§1. What K^nAc Attempted

Through the examination of a series of knowledge acquisition dialogs, one aspect of the knowledge engineer's role in the development of knowledge bases for expert systems became clear: the ability to assimilate information provided by a domain expert into an existing body of knowledge. This task required being able to correlate this new information with existing entity descriptions, recognize the differences between the new and the existing descriptions, and appropriately modify these descriptions based on these differences. The K^nAc system attempted to automate this portion of the knowledge acquisition process.

In order to automate this process, a better understanding of the reasoning used by the knowledge engineer was required. Since this model of the knowledge engineer was likely to be only partially correct and subject to much revision, K^nAc

was designed to make such changes as simple as possible. Thus, this knowledge was encoded as a set of heuristics to which additions, deletions or modifications could be made without requiring other changes to the system. This allows K^nAc to be used as a testbed for refining, through experimentation, a collection of knowledge acquisition techniques.

In addition to isolating the system from the particular set of knowledge acquisition heuristics being used, K^nAc also addressed the issue of how closely tied a knowledge acquisition system must be to the knowledge representation scheme of the target expert system. As the investment in the building of knowledge bases becomes a very significant fraction of the cost of expert system development, the need for reusable knowledge bases becomes ever more apparent. Since a consensus on the ultimate knowledge representation scheme has not yet been reached, knowledge acquisition tools developed for a particular representation will become obsolete with the passing of that representation.

Two factors were considered in order to insulate K^nAc from its target knowledge representation. First, although a knowledge acquisition system must certainly interact with the knowledge base it is modifying, an effort was made to keep the interface between then clean. This does not guarantee that K^nAc can be ported directly to another expert system using a different knowledge representation; there are some dependencies between the representation and the acquisition process. For instance, heuristics developed for one knowledge base may rely on information that a different representation cannot provide. However, by keeping the internal workings of K^nAc separate from those of the knowledge representation, such a transition is made easier.

The second factor considered in designing K^nAc to be applicable to various knowledge representations was the choice of a target representation for its

initial development. The target formalism, as described in Chapter 3, was designed to be fairly general and representative of a large class of frame-based representations. Though this formalism is neither the intersection nor the union of these representations, a dilemma facing the designer of any implementation-independent system, it is intended to contain many of the salient features of this class of representations. Thus, it was intended that this initial system would handle at least some of capabilities provided by most such representations.

Another dimension in which $K^n{}_A c$ is designed to be generic involves the interface between $K^n{}_A c$ and its users. A lesson learned from POISE, a conceptual ancestor of $K^n{}_A c$ in addition to its target expert system, was the distinction between what information is obtained from (or presented to) the end user and the manner in which this information is obtained or presented. In addition to the conceptual clarity this separation provides, it serves to delineate the scope of this project and provides a clean interface to whatever frontend is most appropriate (or available) for a given application. While the example presented herein described the domain expert interacting with $K^n{}_A c$ through a natural language discourse, nothing in the $K^n{}_A c$ system (except for certain knowledge acquisition heuristics) depends on such an interface; a graphic interface (or menu-driven or whatever) would serve just as well.

§2. Where $K^n{}_A c$ Succeeded

This work has made several contributions to the state of automated knowledge acquisition for expert systems. First, a novel approach to knowledge acquisition was proposed and developed. Though a fundamental part of the current conventional knowledge base development process, the issue of automatically lo-

cating and appropriately modifying existing knowledge to conform to the domain experts descriptions has received little, if any, emphasis.

This approach has necessitated a new twist on conventional pattern (or structure) matching techniques. The fairly generic methods described herein for "set" and "constraint" matching are tailored to the knowledge acquisition application. In particular, they have shifted the emphasis from the traditional "How well *do* two structures match?" to "How well *could* they match?".

The likelihood of such matches can only be evaluated in a particular context. If future modifications to the knowledge can be anticipated, the significance of specific match discrepancies can be weighed. This work has developed the concept of "expectations" of such modifications and has shown how various aspects of the knowledge acquisition process may be tapped to produce them.

Specifically, a set of knowledge acquisition heuristics were culled from a series of (human) protocols and were used by K^nA_c to anticipate such modifications. This set, though by no means complete, has been significantly modified and refined as a result of experimentation with the K^nA_c system.

Finally, a demonstration system has been implemented and provides a means of further experimentation. Though the system has shortcomings (see the following section), it has served as a valuable tool for investigating variations to the matching (and match evaluation) process, the form and content of the expectations generated, and the knowledge acquisition heuristics.

§3. Where $K^n{}_Ac$ Failed

Though this work has taken steps towards understanding one aspect of knowledge base development, it has not, unfortunately, solved the knowledge acquisition crisis. Its shortcomings may be divided into two classes: 1) failure of the implementation of the $K^n{}_Ac$ system to provide the functionality proposed by this approach, and 2) failure of this approach, no matter how well implemented, to address the problems posed by knowledge acquisition.

The implementation's shortcomings, while not conceptually significant, are more readily apparent and have served to limit the system's usefulness, even on an experimental basis. For instance, though $K^n{}_Ac$ has been designed to be independent of a particular front-end (i.e., interface to the domain expert), to date, this has been taken *too literally: Currently, there is no* convenient means for interacting with the domain expert. Knowledge acquisition dialogs have been hand-parsed to obtain simulated output from a discourse parser. Even if a front-end existed, the current implementation is too slow to be used in "real-time". (The assimilation of a several sentence "discourse frame" takes several minutes.)

Other issues straddle the boundary between "implementation details" and "conceptual flaws". Consider, for instance, the question of system initiative. In a philosophy shared with its ancestor, POISE, $K^n{}_Ac$ allows a complete spectrum of system vs. user initiative. The system generates expectations of information to be provided by the domain expert. The system is capable of remaining passive and using these expectations to assimilate whatever information is provided. Alternatively, it may use these expectations to interrogate the expert. Just as a (human) knowledge engineer has these options, so has $K^n{}_Ac$. However, this begs the question of deciding the appropriate level of system initiative for a

given situation. Since this problem is at least as difficult as the issues addressed herein, it is summarily dismissed.

On the conceptual side, this work set out to examine a very particular model of the knowledge acquisition process. Specifically, this model assumes that the domain expert presents the domain information to the knowledge engineer, whose task is to integrate this information into a knowledge base. It assumes that the domain expert need not be aware of the form (or even the content) of the target knowledge base and the knowledge engineer need not be familiar with the application domain. The role of the domain expert is to provide information about the domain, not to actively modify the existing knowledge base.

This view was chosen not because it is believed to be the only "correct" approach to knowledge acquisition. Rather, it was selected because 1) it is a reasonable model of what currently occurs, 2) it has not been examined before, and 3) it seems to be a useful component for an overall knowledge acquisition system. It was studied in isolation for pedagogical reasons; the virtues and limits of this approach were better seen by separating it from other knowledge acquisition techniques.

Attempting to replicate some of the knowledge acquisition discourse using this system has proven to be, at times, awkward. While this does not necessarily imply a failure of this approach, either conceptually or as an implementation, it emphasizes that this is only part of the total solution. In practice, the above assumptions are often relaxed. The demarcations between the domain expert, the knowledge engineer and the knowledge base become blurred: The knowledge engineer acquires some familiarity with the application domain; the domain expert infers portions of the (incorrect) model contained in the knowledge base and attempts to modify them. Since not all of the actions taken by the knowledge

engineer (or the domain expert) are easily predicted or explained by this model, the need to integrate this approach with other knowledge acquisition techniques becomes apparent.

§4. Where to from Here?

This work represents a first step towards an intelligent knowledge acquisition system that would assimilate domain information provided by a user into an expert system's knowledge base. Much work remains before such a system is realized. Some of this work involves extensions and improvements to the tool described herein. The integration of this tool with other knowledge acquisition tools is also highly desirable, while the addition of a more practical interface between the user and the $K^n{}_Ac$ system is needed before the system can be truly useful. Finally, the application of the techniques developed for this work to other domains and knowledge representations is being considered.

Extensions to the system itself include: 1) capturing more aspects of the knowledge engineer's expertise through additional knowledge acquisition heuristics; 2) extending the target knowledge representation to more explicitly include formalisms other than "frames", such as production rules and logic; 3) improved efficiency to allow real-time interaction. The integration of this tool with a knowledge base structure editor (i.e., a knowledge acquisition tool that permits direct manipulation of the knowledge structures) should produce a powerful combination. This would combine $K^n{}_Ac$'s knowledge of *what* information to modify with a very straightforward means of specifying *how* to modify it. Other traditional approaches to knowledge acquisition, such as consistency checkers and error-driven systems, could easily augment $K^n{}_Ac$ by serving as additional sources

of "expectations".

Though this system was developed primarily for the construction and modification of knowledge bases, other applications of this technology are possible. For instance, the use of these knowledge acquisition techniques during the running of the underlying expert system in order to dynamically acquire needed information is being considered. Many of the techniques applicable to an explicit knowledge acquisition dialog (e.g., integration of new information, determining knowledge base inconsistencies, etc.) could also be useful for implicitly obtaining and incorporating this "run-time" data.

APPENDIX A

KNOWLEDGE ACQUISITION INTERVIEW

Introduction: This interview of the COINS department's principle clerk concerns the completion of travel vouchers for travel undertaken during the course of studies or research for students or professors at UMass at Amherst. All travel must be work related.

START OF INTERVIEW

Interviewer: So today is November 8th, and we're doing our second interview with Barbara Gould. This interview concerns vouchers for travel, i.e., reimbursement for business expenditures.

Clerk: O.K. – on travel. The proper way of doing it, if it's out of state, is that a travel authorization should be issued before the trip.

——— time frame 1 ———

It can be set up afterwards, but accounting likes to have it in before the trip is taken.

——— time frame 2 ———

And what you do is list: 1) your destination; 2) the date you plan on leaving; 3) the date you are returning; 4) how you plan on going - whether it's plane, bus, private car or whatever; 5) your estimated expenses, and how much of a reimbursement you're getting - whether it's a set amount or whether it's full; 6) the purpose of the trip, and, of course, the account that the money is going to come out of. Then the traveler has to sign that, and, if it comes out of a grant, the P.I. (must sign it); if it's state funds then the department head signs it, but we never get state travel funds. So it had better come out of a grant or a trust fund.

time frame 3

Then once the traveler is back, we have a voucher form that we have to fill out - which gives a detailed account of your trip: 1) the date you left, 2) where you went, like say, you're going to take a plane, then it would be from Amherst to Bradley - there's mileage here - right now it's at 20 cents a mile. Then, let's say, you went to Washington, D.C., you'd list Bradley to Washington. You have to keep receipts for trains, no, sorry for planes and for the hotel. If it were to a conference, then the registration fee, and, if you drove your car own car, you have to have the odometer reading, starting and ending. Meals are a set rate. You don't need to keep receipts for that. Taxis - they don't need receipts - they take your word for it.

time frame 4

When you come back, you submit to whoever's doing the voucher for you, all the receipts you have, and then plan on spending a couple of

minutes with that person. So she can figure your trip out. Go over the trip and its details. So she can get all she needs to know.

time frame 5

The time you leave, the time you came back, determines what kind of meals you are allowed to get. You have to leave at a certain time in order to be entitled to breakfast, and have to come back at a certain time in order to be entitled to supper.

time frame 6

In-state travel is different, in that you don't have to have the travel authorization. So when you come back, you just notify your secretary and she fills out the form.

time frame 7

If you go on a trip of a duration of less than 24 hours, you are not entitled to the lunch. Cause, I guess, they figure, you're going to have lunch anyway. If you leave 2 hours before your regular starting time, you are entitled to breakfast. If you come back 2 hours after your regular ending time, you are entitled to supper.

time frame 8

Oh, your meals right now, depending on your union, there's different rates. The ones like the professors are: $2.50 for breakfast, ah $2.00 for breakfast, $3.00 for dinner, $6.00 for supper.

time frame 9

Your people, who do not have a union, it is now $4.00 for lunch, $7.00 for supper.

time frame 10

You can get a travel advance. There's two ways of doing that.

time frame 11

If it's a professor, and he's getting less than $700, he can go to the bursar's office. Once his travel authorization is filled out and approved, he can collect a cash advance from them.

time frame 12

If you are a student, you can get it directly from the grant that's paying your travel, but you have to give us at least a week's notice in order to get the paperwork processed.

time frame 13

Then the money comes directly from the grant. You'll get a check from your account. So it takes, if you're lucky, you can get it within a week. It really shouldn't take longer than that.

time frame 14

Once you get back, you have to file your voucher. If your advance was more than what you actually spent, then you have to pay the university back what you didn't spend.

——— time frame 15 ———

Interviewer: How about like airplanes? Does the person who's going on the trip make his own airplane reservation?

——— time frame 16 ———

Clerk: I never make any. I think some of the grant secretaries make them for their professors. Janet does it for Ed. I suppose if your boss asks you to, you have to. As a rule, anyone I handle makes his own.

——— time frame 17 ———

Interviewer: So I get the impression, that the person who's going to go on the trip, you know, they contact the P.I., and then they go to the secretary who handles that particular grant, and then that secretary will forward something to the accounting office.

——— time frame 18 ———

Clerk: Right.

Interviewer: O.K.

Clerk: The authorization, then once they're back we do the voucher. We have to do the disbursement schedule and then we do the voucher.

—————— **time frame 19** ——————

Interviewer: Who, does the accounting office, then issue the check to pay the person?

—————— **time frame 20** ——————

Clerk: Right, if they got an advance the check has to go directly to the bursar's office. This is an advancement office.

—————— **time frame 21** ——————

The check, on the disbursement schedule, we address it c/o bursar's office, the check will go to the bursar's office.

—————— **time frame 22** ——————

They will take what's been given as an advance and then send the traveler the remaining money, if it's more. If it's less, then they notify the traveler that they owe money.

—————— **time frame 23** ——————

Interviewer: Does accounting simply make sure that you don't spend more money than you have left, or do they make sure that this is a legitimate expenditure?

—————— **time frame 24** ——————

Clerk: Oh, well, I've never known any authorization to be refused. Supposedly if it's on a grant, then it's supposed to be grant related, but I've never known them to refuse. Just say you're going to a conference and they usually accept it. I suppose if the accountant felt that the trip really had nothing to do with the grant, then they might refuse it, but I've never known that to happen.

———— time frame 25 ————

Interviewer: O.K.

Clerk: As far as trust fund money is concerned, then they have to get approved by Dr. Riseman. Then if he says O.K., you can take it out of the department trust fund. Then it's approved, you know, go ahead and do it.

———— time frame 26 ————

Interviewer: Well, this is a lot easier than the other one. I really like ...

Clerk: Yeah, well, the interview part is easy, but the vouchers can be real tricky, cause people don't do what they say they were going to do, or they put in a little personal trip along side of it.

time frame 27

So you have to explain everything on the voucher. So that when the people over there get a hold of it, they can say "O.K., he didn't charge hotel, cause it was personal", or that his airplane ticket didn't match

with the amount he's getting reimbursed, cause he went someplace in addition and he is just charging the fare that a round trip ticket would normally be to his destination.

time frame 28

Interviewer: So that these forms are pretty much free form, i.e., they allow you to explain what really happened, rather than trying to fit it into slots.

time frame 29

Clerk: Yeah, they want you either to explain on the front or, I usually write a long description on the back. There's one column in here that has to be explained on the back or the front, saying what the expenses were for.

time frame 30

And, of course, your receipts have to be submitted with it. You have to give the original receipts.

time frame 31

Interviewer: O.K.

Clerk: And another thing - say a professor wants to take his wife along, the hotel will be billed for two people. We then need a statement from the hotel stating what the single room rate would be, and that's all they will get reimbursed. They don't get reimbursed for their spouse's half.

time frame 32

Or sometimes, a student, rather a couple of students, will go to to a conference. They will share a room. In that case, we get two bills, or try to get two bills from the hotel stating, you know, and just split it in half.

time frame 33

Sometimes, the hotel will refuse to do that and so in that case, they (accounting) will accept a Xerox copy for one of the vouchers, and you just explain on the back that they shared a room with so-and-so, and that the original is attached to voucher number such-and-such.

time frame 34

But, it's not that difficult, you know it's just a pain in the neck. Well, if they do what they're supposed to do, they're fine, but ...

Interviewer: I think this will be a great example of "O.K., it's never done the way it's supposed to have been done, how are we going to handle this?" Great, I think that's it.

Appendix B

Knowledge Acquisition Heuristics

This appendix contains the knowledge acquisition heuristics currently being used by the $K^n{}_Ac$ system. Each heuristic description tells when the heuristic is applicable, what expectations result when it is invoked, how specific the heuristic is, and its rationale. Note that variables (denoted by **`?variable`** or **`?variable<predicate>`**) contained in a heuristic's *condition* clause are bound when the heuristic is applied; these bindings are used in the generation of the expectations.

§1. State

The heuristics contained in this section use the state of the information in the knowledge base to anticipate modifications.

Heuristic S2

If a field of an entity is determined to contain too few values, additional values (of the appropriate type) will be expected. The expected field size may come, in order of specificity, from meta-information about a given field of a given entity, via inheritance from a generalization of the entity, from the default information for class of the entity, or from an overall field default size. This size information may be static or determined dynamically by the system.

```
H_S2 is a MODERATE specificity, STATE heuristic.
Its rationale is:
    "Fields with too few components will be augmented."

It is triggered by:
   ENTITY STATE: INSUFFICIENT-VALUES (certainty ?CERTAINTY) for
                    entity ?ENTITY<IS-A-KNAC-STRUCTURE-P>.
                 Field: ?FIELD   Values: ?VALUES

It results in:
  (ADD-EXPECTATION
      :MODIFICATION
         (MAKE-MODIFICATION
           :ACTION-TYPE 'ADD
           :FIELD '?FIELD
           :VALUE '?NEW-PART<IS-A-KNAC-STRUCTURE-P>
           :TARGET '?ENTITY)
      :EFFECTIVE-TIME-FRAME 'FADE
      :CERTAINTY '?CERTAINTY)
```

Heuristic S3

Each event may have a *pre-condition* and a *post-condition* that must be satisfied when the event is begun and finished, respectively. Each step in an event, being itself an event, may have a *pre-* and *post-condition*. A step's *pre-condition* may be satisfied either by the *post-condition* of an earlier step in the plan or may be a *pre-condition* of the (parent) plan. When neither of these situations can be deduced, it may imply that a step is missing.

```
H_S3 is a MODERATE specificity, STATE heuristic.
Its rationale is:
    "Unsatisfied step preconditions will be satisfied."

It is triggered by:
   ENTITY STATE: PRECONDITION-IS-NOT-SATISFIED
                    (certainty ?CERTAINTY) for
                    entity ?ENTITY<IS-AN-EVENT-P>.
```

```
                    Field: PARTS   Values: ?VALUES

It results in:
  '(,(ADD-EXPECTATION
          :MODIFICATION
              (MAKE-MODIFICATION
                :ACTION-TYPE 'ADD
                :FIELD 'PARTS
                :VALUE '?NEW-STEP<IS-AN-EVENT-P>
                :TARGET'?ENTITY)
          :EFFECTIVE-TIME-FRAME 'FADE
          :CERTAINTY '?CERTAINTY)
    ,(ADD-EXPECTATION
          :MODIFICATION
              (MAKE-MODIFICATION
                :ACTION-TYPE 'ADD
                :FIELD 'CONSTRAINTS
                :VALUE '(BEFORE ?NEW-STEP<IS-AN-EVENT-P>
                                ,(FIRST '?VALUES))
                :TARGET '?ENTITY)
          :EFFECTIVE-TIME-FRAME 'FADE
          :CERTAINTY '?CERTAINTY))
```

Heuristic S4

Type-checking is provided by most knowledge bases. If the value of an attribute is not in its permitted range, for example, a modification may be expected.

```
H_S4 is a SPECIFIC specificity, STATE heuristic.
Its rationale is:
    "Attribute values must be within specified ranges."

It is triggered by:
   ENTITY STATE: VALUE-RANGE-CONFLICT (certainty ?CERTAINTY)
                  for  entity ?ENTITY<IS-A-KNAC-STRUCTURE-P>.
                 Field: ATTRIBUTE-VALUES
                 Values: (?ATTRIBUTE ?VALUE ?RANGE)

It results in:
   (ADD-EXPECTATION
```

```
:MODIFICATION
   (MAKE-MODIFICATION
      :ACTION-TYPE 'REMOVE
      :FIELD 'ATTRIBUTE-VALUES
      :VALUE '(?ATTRIBUTE ?VALUE))
:EFFECTIVE-TIME-FRAME
   '(UNTIL IS-A?
           (GET-ATTRIBUTE-VALUE ?ENTITY ?ATTRIBUTE)
           (GET-ATTRIBUTE-RANGE ?ENTITY ?ATTRIBUTE))
:CERTAINTY '?CERTAINTY)
```

Heuristic S6

If the user refers to an entity not contained in the knowledge base, the addition of this entity is expected.

```
H_S6 is a SPECIFIC specificity, STATE heuristic.
Its rationale is:
    "Referenced entities should exist."

It is triggered by:
   ENTITY STATE: UNBOUND-REFERENCE (certainty ?CERTAINTY)
                   for entity ?ENTITY<IS-A-KNAC-STRUCTURE-P>.
                 Field: ?FIELD
                 Values: (?NEW-ENTITY ?ENTITY-TYPE)

It results in:
  (ADD-EXPECTATION
      :MODIFICATION
         (MAKE-MODIFICATION :ACTION-TYPE 'CREATE
                            :FIELD ()
                            :VALUE '?ENTITY-TYPE
                            :TARGET '?NEW-ENTITY)
      :EFFECTIVE-TIME-FRAME 'FADE
      :CERTAINTY '?CERTAINTY)
```

§2. Modifications

A series of modifications to the knowledge base are often related. Hence, the occurrence of one such modification may result in the expectation of others. The heuristics in this section capture this intuition.

Heuristic M1

After creating a new entity, the addition of its subparts, attributes and specializations may be expected.

```
H_M1 is a MODERATE specificity, MODIFICATION heuristic.
Its rationale is:
    "Detailed information usually follows the
     introduction of a new entity."

It is triggered by:
   MOD1: CREATE the (?Entity-type) ?Entity1

It results in:
  (LET ((TARGET-TYPE (TYPE-OF (STRUCTURE-EVAL '?ENTITY1))))
     (MAPC-CONDCONS
        #'(LAMBDA (FIELD VALUE-PREDICATE)
             (ADD-EXPECTATION
                  :MODIFICATION
                    (MAKE-MODIFICATION
                       :ACTION-TYPE 'ADD
                        :FIELD FIELD
                        :VALUE
                           (MAKE-PATTERN-VAR
                              :NAME 'NEW-VALUE
                              :PREDICATE VALUE-PREDICATE)
                        :TARGET '?ENTITY1)
                  :EFFECTIVE-TIME-FRAME 'FADE
                  :CERTAINTY '?RATING))
        '(PARTS SPECIALIZATIONS ATTRIBUTES)
        '((LAMBDA (VAL)
             (IS-A? VAL ',TARGET-TYPE))
          (LAMBDA (VAL)
```

```
      (IS-A? VAL ',TARGET-TYPE))
    IS-A-KNAC-STRUCTURE-P)))
```

Heuristic M2

An entity is usually hooked into those entities of which it is a part or into appropriate generalizations.

```
H_M2 is a MODERATE specificity, MODIFICATION heuristic.
Its rationale is:
    "Context information usually follows the
     introduction of a new entity."

It is triggered by:
   MOD2: CREATE the (?Entity-type) ?Entity1

It results in:
  (LET ((TARGET-TYPE (TYPE-OF (STRUCTURE-EVAL '?ENTITY1))))
    (MAPC-CONDCONS
     #'(LAMBDA (FIELD)
         (ADD-EXPECTATION
              :MODIFICATION
                (MAKE-MODIFICATION
                   :ACTION-TYPE 'ADD
                   :FIELD FIELD
                   :VALUE
   (MAKE-PATTERN-VAR
     :NAME 'NEW-VALUE
     :PREDICATE
       `(LAMBDA (VAL)
          (IS-A? VAL ',TARGET-TYPE)))
                   :TARGET '?ENTITY1)
              :EFFECTIVE-TIME-FRAME 'FADE
              :CERTAINTY '?RATING))
     '(GENERALIZATIONS PART-OF)))
```

Heuristic M3

Constraints on the range of an attribute or its relations to other attributes are often provided after the attribute is added.

```
H_M3 is a MODERATE specificity, MODIFICATION heuristic.
Its rationale is:
    "Attributes are usually constrained after
     being introduced."

It is triggered by:
   MOD3: ADD ?Attribute to the Attribute-names field
         of ?Entity1<is-a-knac-structure-p>

It results in:
  (ADD-EXPECTATION
      :MODIFICATION
        (MAKE-MODIFICATION
          :ACTION-TYPE 'ADD
          :FIELD 'CONSTRAINTS
          :VALUE
            `(,(GENVAR RELATION IS-A-RELATIONSHIP-P) ?ATTRIBUTE
               ,(GENVAR ATTRIBUTE IS-A-KNAC-STRUCTURE-P))
          :TARGET '?ENTITY1)
      :EFFECTIVE-TIME-FRAME 'FADE
      :CERTAINTY '?RATING)
```

Heuristic M4

The relationships of a new part of an entity to existing parts are often added following the addition of the part. This includes temporal relationships between steps of an Event and spatial relationships between parts of an Object.

```
H_M4 is a MODERATE specificity, MODIFICATION heuristic.
Its rationale is:
    "Parts are usually constrained after being introduced."

It is triggered by:
   MOD4: ADD ?Part1 to the Parts field
         of ?Entity1<is-a-knac-structure-p>
```

```
It results in:
   (LET ((PART-TYPE (TYPE-OF (STRUCTURE-EVAL '?ENTITY1))))
     (ADD-EXPECTATION
          :MODIFICATION
            (MAKE-MODIFICATION
              :ACTION-TYPE 'ADD
              :FIELD 'CONSTRAINTS
              :VALUE
                `(?RELATION<IS-A-RELATIONSHIP-P>
                     ?PART1
                     ,(MAKE-PATTERN-VAR
                        :NAME 'PART2
                        :PREDICATE
                          `(LAMBDA (PART)
                             (IS-A? PART ',PART-TYPE))))
               :TARGET '?ENTITY1)
          :EFFECTIVE-TIME-FRAME 'FADE
          :CERTAINTY '?RATING))
```

Heuristic M4a

This is an Event-specific version of heuristic M4.

```
H_M4A is a SPECIFIC specificity, MODIFICATION heuristic.
Its rationale is:
     "Parts of Events are usually temporally
      constrained after being introduced."

It is triggered by:
   MOD5: ADD ?Step1 to the Parts field
         of ?Event1<is-an-event-p>

It results in:
  (ADD-EXPECTATION
       :MODIFICATION
         (MAKE-MODIFICATION
           :ACTION-TYPE 'ADD
           :FIELD 'CONSTRAINTS
           :VALUE
             `(?TEMPORAL-RELATION<IS-A-TEMPORAL-RELATIONSHIP-P>
                 ?STEP1 ?STEP2<IS-AN-EVENT-P>)
```

```
        :TARGET '?EVENT1)
    :EFFECTIVE-TIME-FRAME 'FADE
    :CERTAINTY '?RATING)
```

Heuristic M4b

This is an Object-specific version of heuristic M4.

```
H_M4B is a SPECIFIC specificity, MODIFICATION heuristic.
Its rationale is:
    "Parts of Objects are usually spatially
     constrained after being introduced."

It is triggered by:
   MOD6: ADD ?Piece1 to the Parts field of
          ?Object1<is-an-object-p>

It results in:
   (ADD-EXPECTATION
        :MODIFICATION
          (MAKE-MODIFICATION
             :ACTION-TYPE 'ADD
             :FIELD 'CONSTRAINTS
             :VALUE
               `(?SPATIAL-RELATION<IS-A-SPATIAL-RELATIONSHIP-P>
                    ?PIECE1 ?PIECE2<IS-AN-OBJECT-P>)
             :TARGET '?OBJECT1)
        :EFFECTIVE-TIME-FRAME 'FADE
        :CERTAINTY '?RATING)
```

Heuristic M5

This is a more constrained version of heuristic M4.

```
H_M5 is a MODERATE specificity, MODIFICATION heuristic.
Its rationale is:
    "Adding two parts to an entity usually
     implies a relation between them."
```

```
It is triggered by:
   (AND
     MOD7: ADD ?Part1 to the Parts field
           of ?Entity1<is-a-knac-structure-p>
     MOD8: ADD ?Part2 to the Parts field
           of ?Entity1<is-a-knac-structure-p> )

It results in:
   (ADD-EXPECTATION
       :MODIFICATION
         (MAKE-MODIFICATION
           :ACTION-TYPE 'ADD
           :FIELD 'CONSTRAINTS
           :VALUE
             '(?RELATION1<IS-A-KNAC-RELATION-P>
                 ?PART1 ?PART2)
           :TARGET '?ENTITY1)
       :EFFECTIVE-TIME-FRAME 'FADE
       :CERTAINTY '?RATING)
```

Heuristic M5a

This is an Event-specific version of heuristic M5.

```
H_M5A is a SPECIFIC specificity, MODIFICATION heuristic.
Its rationale is:
    "Adding two steps to an event usually implies
     a temporal relation between these steps."

It is triggered by:
   (AND
     MOD9: ADD ?Step1 to the Parts field
           of ?Event1<is-an-event-p>
     MOD10: ADD ?Step2 to the Parts field
            of ?Event1<is-an-event-p> )

It results in:
   (ADD-EXPECTATION
       :MODIFICATION
         (MAKE-MODIFICATION
           :ACTION-TYPE 'ADD
```

```
        :FIELD 'CONSTRAINTS
        :VALUE
         '(?TEMPORAL-RELATION1<IS-A-TEMPORAL-RELATIONSHIP-P>
             ?STEP1 ?STEP2)
        :TARGET '?EVENT1)
    :EFFECTIVE-TIME-FRAME 'FADE
    :CERTAINTY '?RATING)
```

Heuristic M5b

This is an Object-specific version of heuristic M5.

```
H_M5B is a SPECIFIC specificity, MODIFICATION heuristic.
Its rationale is:
    "Adding two parts to an object usually implies
     a spatial relation between these steps."

It is triggered by:
   (AND
     MOD11: ADD ?Piece1 to the Parts field
            of ?Object1<is-an-object-p>
     MOD12: ADD ?Piece2 to the Parts field
            of ?Object1<is-an-object-p> )

It results in:
   (ADD-EXPECTATION
       :MODIFICATION
         (MAKE-MODIFICATION
           :ACTION-TYPE 'ADD
           :FIELD 'CONSTRAINTS
           :VALUE
             '(?SPATIAL-RELATION1<IS-A-SPATIAL-RELATIONSHIP-P>
                 ?PIECE1 ?PIECE2)
           :TARGET '?OBJECT1)
       :EFFECTIVE-TIME-FRAME 'FADE
       :CERTAINTY '?RATING)
```

Heuristic M6

Any entity containing a reference to a deleted entity may expect a modification to its field that contains this reference.

```
H_M6 is a MODERATE specificity, MODIFICATION heuristic.
Its rationale is:
    "Pointers to an Entity usually change if the
     entity is deleted."

It is triggered by:
   MOD13: DELETE ?Entity1<is-a-knac-structure-p>

It results in:
   (ADD-EXPECTATION
      :MODIFICATION
        (MAKE-MODIFICATION
           :ACTION-TYPE 'REMOVE
           :VALUE '?ENTITY1
           :TARGET '?ENTITY2<IS-A-KNAC-STRUCTURE-P>
           :FIELD '?FIELD)
      :EFFECTIVE-TIME-FRAME 'FADE
      :CERTAINTY '?RATING)
```

Heuristic M7

Since several modifications to an entity often occur together, one modification to a particular entity makes that entity a likely candidate for other modifications. Basically, this is the same principle used in caching.

```
H_M7 is a VAGUE specificity, MODIFICATION heuristic.
Its rationale is:
    "Recently modified entities may be
     modified again or referenced."

It is triggered by:
   MOD14: ?ACTION1<(LAMBDA (ACTION)
                     (NOT (EQL ACTION 'DELETE)))>
          ?Value1 to/from the ?Field1 field
          of ?Entity1<is-a-knac-structure-p>
```

It results in:

```
(CONDCONS
      (ADD-EXPECTATION
                  :MODIFICATION
                    (MAKE-MODIFICATION
                      :ACTION-TYPE '?ACTION2
                      :TARGET '?ENTITY1
                      :FIELD '?FIELD2
                      :VALUE '?VALUE2)
                  :EFFECTIVE-TIME-FRAME 'FADE
                  :CERTAINTY '?RATING)
  (CONDCONS
      (ADD-EXPECTATION
                  :MODIFICATION
                    (MAKE-MODIFICATION
                      :ACTION-TYPE '?ACTION2
                      :TARGET '?ENTITY2
                      :FIELD '?FIELD2
                      :VALUE '?ENTITY1)
                  :EFFECTIVE-TIME-FRAME 'FADE
                  :CERTAINTY '?RATING)
   ())) .
```

§3. Discourse

The discourse between the domain expert and the knowledge engineer (or K^nAc), whether it be via "natural language" or another type of interface, provides certain cues from which information may be anticipated. The more sophisticated the ***discourse manager***, the larger the set of such cues that it can detect. Since K^nAc's current interface is quite primitive, the current set of discourse heuristics is minimal.

Heuristic D1

If the discourse manager is able to determine the topics of a discourse, the portions of the knowledge base most likely to be of interest are probably near these topics.

```
H_D1 is a VAGUE specificity, DISCOURSE heuristic.
Its rationale is:
    "Entities close to specified topics
     are likely to be referenced or modified."

It is triggered by:
   ?TOPIC<IS-A-KNAC-STRUCTURE-P>

It results in:
   (MAPC-APPEND
     #'(LAMBDA (EACH-CLOSE-ENTITY-INFO)
          (LET* ((EACH-CLOSE-ENTITY
                    (FIRST EACH-CLOSE-ENTITY-INFO))
            (DISTANCE (REST EACH-CLOSE-ENTITY-INFO))
            (MAX-DISTANCE (1+ *DEFAULT-SEARCH-DISTANCE*))
            (ENTITY-CERTAINTY
              (/ (- MAX-DISTANCE DISTANCE) MAX-DISTANCE))
            (EXP-1
              (ADD-EXPECTATION
                :MODIFICATION
                  (MAKE-MODIFICATION
                    :ACTION-TYPE '?ACTION
                    :FIELD '?FIELD
                    :VALUE '?VALUE
                    :TARGET EACH-CLOSE-ENTITY)
                :EFFECTIVE-TIME-FRAME
                  '(WHILE MEMBER '?TOPIC *TOPICS*)
                :CERTAINTY ENTITY-CERTAINTY))
            (EXP-2
              (ADD-EXPECTATION
                :MODIFICATION
                  (MAKE-MODIFICATION
                    :ACTION-TYPE '?ACTION
                    :FIELD '?FIELD
                    :VALUE EACH-CLOSE-ENTITY
                    :TARGET '?TARGET<IS-A-KNAC-STRUCTURE-P>)
```

```
                    :EFFECTIVE-TIME-FRAME
                      '(WHILE MEMBER '?TOPIC *TOPICS*)
                    :CERTAINTY ENTITY-CERTAINTY)))
              (CONDCONS EXP-1 (CONDCONS EXP-2 ()))))
         (FIND-CLOSE-ENTITIES '?TOPIC))
```

Heuristic D2

There is generally some degree of continuity in a discourse. An entity is usually not referenced and then never mentioned again. Thus, referenced entities are likely candidates for future reference or modification.

```
H_D2 is a VAGUE specificity, DISCOURSE heuristic.
Its rationale is:
    "Referenced entities are likely
     to be modified or referenced again."

It is triggered by:
   ?DISCOURSE-MATCH<(LAMBDA (VAR)
                       (TYPEP VAR 'DISCOURSE-MATCH))>

It results in:
   (LET ((KB-ENTITY
            (DISCOURSE-MATCH-KB-ENTITY ?DISCOURSE-MATCH))
         (MATCH-RATING
            (DISCOURSE-MATCH-RATING ?DISCOURSE-MATCH)))
     (WHEN KB-ENTITY
       (MAPC-APPEND
         #'(LAMBDA (EACH-CLOSE-ENTITY-INFO)
            (LET* ((EACH-CLOSE-ENTITY
                       (FIRST EACH-CLOSE-ENTITY-INFO))
                   (DISTANCE (REST EACH-CLOSE-ENTITY-INFO))
                   (MAX-DISTANCE
                      (1+ *DEFAULT-SEARCH-DISTANCE*))
                   (ENTITY-RATING
                      (/ (- MAX-DISTANCE DISTANCE)
                         MAX-DISTANCE))
                   (CERTAINTY (COMBINE-CERTAINTIES
                                 MATCH-RATING ENTITY-RATING))
                   (EXP-1
                     (ADD-EXPECTATION
```

```
          :MODIFICATION
            (MAKE-MODIFICATION
              :ACTION-TYPE '?ACTION
              :TARGET EACH-CLOSE-ENTITY
              :FIELD '?FIELD
              :VALUE '?VALUE)
          :EFFECTIVE-TIME-FRAME 'FADE
          :CERTAINTY CERTAINTY))
      (EXP-2
        (ADD-EXPECTATION
          :MODIFICATION
            (MAKE-MODIFICATION
              :ACTION-TYPE '?ACTION
              :TARGET ?TARGET
              :FIELD '?FIELD
              :VALUE EACH-CLOSE-ENTITY)
          :EFFECTIVE-TIME-FRAME 'FADE
          :CERTAINTY CERTAINTY)))
  (CONDCONS EXP-1 (CONDCONS EXP-2 ()))))
(FIND-CLOSE-ENTITIES KB-ENTITY))))
```

APPENDIX C

SAMPLE $K^{N}{}_{A}C$ RESULTS

This appendix contains the results of running the $K^{n}{}_{A}c$ system on the discourse shown in Appendix A. For each discourse frame, a portion of the discourse was parsed, by hand, into the descriptions shown. These descriptions were compared to the indicated candidate knowledge base structures. Where no similar structures are found, the new descriptions were added to the knowledge base. Where sufficiently similar structures were identified, the ensuing modifications were made.

§1. FRAME-1

```
 Interviewer:  So today is November 8th, and we're doing our second
               interview with Barbara Gould.  This interview
               concerns vouchers for travel, i.e., reimbursement
               for business reimbursement for business
               expenditures.

    Clerk:     O.K.  -- on travel.  The proper way of doing it, if
               it's out of state, is that a travel authorization
               should be issued before the trip.

   ------ Discourse structures -------

{ EVENT:  Event-1

   CONSTRAINTS: ( (EQUAL ACTOR ISSUE_TRAVEL_AUTHORIZATION.ISSUEE)
                  (EQUAL DESTINATIONS TAKE_A_TRIP.DESTINATIONS)
                  (BEFORE ISSUE_TRAVEL_AUTHORIZATION TAKE_A_TRIP)
                  (EQUAL ISSUE_TRAVEL_AUTHORIZATION.ISSUEE
                         TAKE_A_TRIP.TRAVELER)
```

```
                    (OUTSIDE DESTINATION STATE) )

   ATTRIBUTES: ( (DESTINATIONS TAKE_A_TRIP.DESTINATIONS)
                 (ACTOR ISSUE_TRAVEL_AUTHORIZATION.ISSUEE) )

   PARTS: ( ISSUE_TRAVEL_AUTHORIZATION
            TAKE_A_TRIP )

   GENERALIZATIONS: ( EVENT )

   TEMPORAL-RELATIONSHIPS: ( (ISSUE_TRAVEL_AUTHORIZATION BEFORE
                              TAKE_A_TRIP) ) }

{ OBJECT:  Travel-Authorization

   PARTS: ( GRANT
            DESTINATION
            DEPARTURE-DATE
            RETURN-DATE
            TRUST-FUND
            STATE-FUNDS
            DEPARTMENT-HEAD
            P-I
            MEANS-OF-TRANSPORT )

   GENERALIZATIONS: ( FORM )

   ASSOCIATED-EVENTS: ( FILL_OUT_TRAVEL_AUTHORIZATION
                        SIGN_FORM<1>
                        SIGN_FORM<2>
                        SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING ) }

{ EVENT:  Take_A_Trip

   ATTRIBUTES: ( (DESTINATION NIL (RANGE . LOCATION))
                 (TRAVELER NIL (RANGE . PERSON)) )

   PART-OF: ( EVENT-3
              EVENT-1 )

   GENERALIZATIONS: ( EVENT ) }
```

```
{ EVENT:  Issue_Travel_Authorization

   ATTRIBUTES: ( (ISSUER NIL (RANGE . PERSON))
                 (ISSUEE NIL (RANGE . PERSON))
                 (ISSUE-TIME NIL (RANGE . TIME)) )

   PART-OF: ( EVENT-1 )

   GENERALIZATIONS: ( ISSUE )

   ASSOCIATED-OBJECTS: ( TRAVEL-AUTHORIZATION ) }

     ------ KB Candidates -------

0.300  TRAVEL
0.240  TAKE_A_TRIP
0.200  DESTINATION
0.200  SOURCE
0.200  MAKE_A_RESERVATION
0.200  PAY
0.200  GO_SOMEWHERE
0.200  DRIVE
0.200  FLY
0.200  TRAVELER
0.200  EVENT
0.196  TAKE_A_TRIP_AND_GET_PAID
0.196  VISIT

     ------ Selected matches -------

TAKE_A_TRIP already in KB

No match found for ISSUE_TRAVEL_AUTHORIZATION

No match found for TRAVEL-AUTHORIZATION

EVENT-1 matches TAKE_A_TRIP_AND_GET_PAID (0.285)
        closest:
           0.27 0.30  TAKE_A_TRIP_AND_GET_PAID
           0.27 0.20  VISIT
           0.24 0.17  MAKE_A_RESERVATION
           0.24 0.17  TRAVEL
```

------ Performed modifications -------

MOD174: CREATE the event Issue_travel_authorization

MOD175: ADD Issue-time to the Attribute-names field
of Issue_travel_authorization

MOD176: ADD Issuee to the Attribute-names field
of Issue_travel_authorization

MOD177: ADD Issuer to the Attribute-names field
of Issue_travel_authorization

MOD178: ADD (Issuer nil (range . person)) to the Attributes
field of Issue_travel_authorization

MOD179: ADD (Issuee nil (range . person)) to the Attributes
field of Issue_travel_authorization

MOD180: ADD (Issue-time nil (range . time)) to the Attributes
field of Issue_travel_authorization

MOD181: ADD Event-1 to the Part-of field
of Issue_travel_authorization

MOD182: ADD Issue to the Generalizations field
of Issue_travel_authorization

MOD183: ADD Travel-authorization to the Associated-objects
field of Issue_travel_authorization

MOD531: CREATE the object Travel-authorization

MOD532: ADD Form to the Generalizations field
of Travel-authorization

MOD533: ADD Issue_travel_authorization to the Associated-events
field of Travel-authorization

MOD1154: ADD Issue_travel_authorization to the Parts
field of Take_a_trip_and_get_paid

```
        Mod certainty: 0.036
        Expected: 0.332(avg)  0.480(max) 0.719(ind)

MOD1156: ADD Actor to the Attribute-names field of
             Take_a_trip_and_get_paid
         Mod certainty: 0.002
         Expected: 0.157(avg)  0.157(max)  0.157(ind)

MOD1262: ADD (Outside destination state) to the
             Constraints field of Take_a_trip_and_get_paid
         Mod certainty: 0.048

MOD1263: ADD (Before issue_travel_authorization take_a_trip
             to the Constraints field of Take_a_trip_and_get_paid
         Mod certainty: 0.048

MOD1264: ADD (Equal actor issue_travel_authorization.issuee) to
             the Constraints field of Take_a_trip_and_get_paid
         Mod certainty: 0.048

MOD1265: ADD (Equal issue_travel_authorization.issuee
                     get_reimbursed.recipient)
             to the Constraints field of Take_a_trip_and_get_paid
         Mod certainty: 0.048
```

§2. FRAME-2

```
Clerk:  It can be set up afterwards, but accounting likes to
        have it in before the trip is taken.

    ------ Discourse structures -------

{ OBJECT:  Accounting

   GENERALIZATIONS: ( DEPARTMENT )

   ASSOCIATED-EVENTS: ( SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING ) }

{ OBJECT:  Travel-Authorization
```

```
   PARTS: ( GRANT
            DESTINATION
            DEPARTURE-DATE
            RETURN-DATE
            TRUST-FUND
            STATE-FUNDS
            DEPARTMENT-HEAD
            P-I
            MEANS-OF-TRANSPORT )

   GENERALIZATIONS: ( FORM )

   ASSOCIATED-EVENTS: ( FILL_OUT_TRAVEL_AUTHORIZATION
                        SIGN_FORM<1>
                        SIGN_FORM<2>
                        SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING ) }

{ EVENT:  Send_Travel_Authorization_To_Accounting

   CONSTRAINTS: ( (EQUAL INFORMATION 'TRAVEL-AUTHORIZATION)
                  (EQUAL RECIPIENT 'ACCOUNTING) )

   ATTRIBUTES: ( (SENDER NIL (RANGE . PERSON))
                 (RECIPIENT 'ACCOUNTING)
                 (INFORMATION 'TRAVEL-AUTHORIZATION) )

   GENERALIZATIONS: ( SEND_INFORMATION )

   ASSOCIATED-OBJECTS: ( TRAVEL-AUTHORIZATION
                         ACCOUNTING ) }

     ------ KB Candidates -------

0.480  TRAVEL-AUTHORIZATION
0.480  ISSUE_TRAVEL_AUTHORIZATION
0.384  VISIT
0.384  EVENT
0.384  TRAVELER
0.384  FLY
0.384  DRIVE
0.384  GO_SOMEWHERE
0.384  PAY
```

```
0.384  MAKE_A_RESERVATION
0.384  SOURCE
0.384  DESTINATION
0.384  TAKE_A_TRIP_AND_GET_PAID
0.384  TAKE_A_TRIP
0.288  TRAVEL
0.240  ACCOUNTING
0.196  UNIVERSITY
0.196  DEPARTMENT
0.196  OBJECT
0.196  FORM

      ------ Selected matches -------

TRAVEL-AUTHORIZATION ACCOUNTING already in KB

No match found for SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING

      ------ Performed modifications -------

 MOD1555: CREATE the event Send_travel_authorization_to_accounting

 MOD1556: ADD (Equal information 'travel-authorization) to the
              Constraints field of
              Send_travel_authorization_to_accounting

 MOD1557: ADD (Equal recipient 'accounting) to the Constraints
              field of Send_travel_authorization_to_accounting

 MOD1558: ADD Information to the Attribute-names field of
              Send_travel_authorization_to_accounting

 MOD1559: ADD Recipient to the Attribute-names field of
              Send_travel_authorization_to_accounting

 MOD1560: ADD Sender to the Attribute-names field of
              Send_travel_authorization_to_accounting

 MOD1561: ADD (Sender nil (range . person)) to the Attributes field
              of Send_travel_authorization_to_accounting

 MOD1562: ADD (Recipient 'accounting) to the Attributes field of
```

Send_travel_authorization_to_accounting

MOD1563: ADD (Information 'travel-authorization) to the Attributes field of Send_travel_authorization_to_accounting

MOD1564: ADD Send_information to the Generalizations field of Send_travel_authorization_to_accounting

MOD1565: ADD Travel-authorization to the Associated-objects field of Send_travel_authorization_to_accounting
Expected: 0.192(avg) 0.192(max) 0.656(ind)

MOD1566: ADD Accounting to the Associated-objects field of Send_travel_authorization_to_accounting
Expected: 0.192(avg) 0.192(max) 0.192(ind)

§3. FRAME-3

Clerk: And what you do is list: 1) your destination; 2) the date you plan on leaving; 3) the date you are returning; 4) how you plan on going - whether it's plane, bus, private car or whatever; 5) your estimated expenses, and how much of a reimbursement you're getting - whether it's a set amount or whether it's full; 6) the purpose of the trip, and, of course, the account that the money is going to come out of. Then the traveler has to sign that, and, if it comes out of a grant, the P.I. (*must sign it*); if it's state funds then the department head signs it, but we never get state travel funds. So it better come out of a grant or a trust fund.

------ Discourse structures -------

{ EVENT: Sign_Form

CONSTRAINTS: ((EQUAL SIGNEE 'TRAVELER)
(EQUAL SIGNEE 'P.I.))

ATTRIBUTES: ((SIGNEE 'TRAVELER)

```
                (SIGNEE 'P.I.) )

   PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )

   GENERALIZATIONS: ( EVENT )

   ASSOCIATED-OBJECTS: ( TRAVEL-AUTHORIZATION ) }

{ OBJECT:  Private-Car

   ATTRIBUTES: ( (START-ODOMETER-READING NIL (RANGE . NUMBER))
                (END-ODOMETER-READING NIL (RANGE . NUMBER)) )

   GENERALIZATIONS: ( OBJECT ) }

{ OBJECT:  Bus

   GENERALIZATIONS: ( MEANS-OF-TRANSPORT ) }

{ OBJECT:  Plane

   GENERALIZATIONS: ( MEANS-OF-TRANSPORT ) }

{ OBJECT:  Means-Of-Transport

   PART-OF: ( TRAVEL-AUTHORIZATION )

   SPECIALIZATIONS: ( TAXI
                      PLANE
                      BUS
                      PRIVATE-CAR )

   GENERALIZATIONS: ( OBJECT ) }

{ OBJECT:  P-I

   PART-OF: ( TRAVEL-AUTHORIZATION )

   GENERALIZATIONS: ( PERSON ) }

{ OBJECT:  Department-Head
```

```
   PART-OF: ( TRAVEL-AUTHORIZATION )
   GENERALIZATIONS: ( PERSON ) }
{ OBJECT:  State-Funds
   PART-OF: ( TRAVEL-AUTHORIZATION )
   GENERALIZATIONS: ( MONEY ) }
{ OBJECT:  Trust-Fund
   PART-OF: ( TRAVEL-AUTHORIZATION )
   GENERALIZATIONS: ( MONEY ) }
{ OBJECT:  Return-Date
   PART-OF: ( TRAVEL-AUTHORIZATION )
   GENERALIZATIONS: ( DATE ) }
{ OBJECT:  Departure-Date
   PART-OF: ( TRAVEL-VOUCHER )
   GENERALIZATIONS: ( DATE ) }
{ OBJECT:  Destination
   PART-OF: ( TRAVEL-VOUCHER )
   GENERALIZATIONS: ( LOCATION ) }
{ OBJECT:  Grant
   PART-OF: ( TRAVEL-AUTHORIZATION )
   GENERALIZATIONS: ( MONEY ) }
{ EVENT:  Sign_Form<2>
```

```
    CONSTRAINTS: ( (EQUAL SIGNEE 'P.I.) )
    ATTRIBUTES: ( (SIGNEE 'P.I.) )
    PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )
    GENERALIZATIONS: ( EVENT )
    ASSOCIATED-OBJECTS: ( TRAVEL-AUTHORIZATION ) }
{ EVENT:  Sign_Form<1>
    CONSTRAINTS: ( (EQUAL SIGNEE 'TRAVELER) )
    ATTRIBUTES: ( (SIGNEE 'TRAVELER) )
    PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )
    GENERALIZATIONS: ( EVENT )
    ASSOCIATED-OBJECTS: ( TRAVEL-AUTHORIZATION ) }
{ EVENT:  Fill_Out_Form_Field<7>
    CONSTRAINTS: ( (EQUAL FIELD 'PURPOSE) )
    ATTRIBUTES: ( (FIELD 'PURPOSE) )
    PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )
    GENERALIZATIONS: ( EVENT ) }
{ EVENT:  Fill_Out_Form_Field<6>
    CONSTRAINTS: ( (EQUAL FIELD 'REIMBURSEMENT) )
    ATTRIBUTES: ( (FIELD 'REIMBURSEMENT) )
    PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )
    GENERALIZATIONS: ( EVENT ) }
```

```
{ EVENT:  Fill_Out_Form_Field<5>

   CONSTRAINTS: ( (EQUAL FIELD 'ESTIMATED-EXPENSES) )

   ATTRIBUTES: ( (FIELD 'ESTIMATED-EXPENSES) )

   PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )

   GENERALIZATIONS: ( EVENT ) }

{ EVENT:  Fill_Out_Form_Field<4>

   CONSTRAINTS: ( (EQUAL FIELD 'MEANS-OF-TRAVEL) )

   ATTRIBUTES: ( (FIELD 'MEANS-OF-TRAVEL) )

   PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )

   GENERALIZATIONS: ( EVENT ) }

{ EVENT:  Fill_Out_Form_Field<3>

   CONSTRAINTS: ( (EQUAL FIELD 'RETURN-DATE) )

   ATTRIBUTES: ( (FIELD 'RETURN-DATE) )

   PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )

   GENERALIZATIONS: ( EVENT ) }

{ EVENT:  Fill_Out_Form_Field<2>

   CONSTRAINTS: ( (EQUAL FIELD 'DEPARTURE-DATE) )

   ATTRIBUTES: ( (FIELD 'DEPARTURE-DATE) )

   PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )

   GENERALIZATIONS: ( EVENT ) }

{ EVENT:  Fill_Out_Form_Field<1>
```

```
   CONSTRAINTS: ( (EQUAL FIELD 'DESTINATION) )

   ATTRIBUTES: ( (FIELD 'DESTINATION) )

   PART-OF: ( FILL_OUT_TRAVEL_AUTHORIZATION )

   GENERALIZATIONS: ( EVENT ) }

{ EVENT:  Fill_Out_Travel_Authorization

   CONSTRAINTS: ( (SAME_AS FORM-FILLER 'TRAVELER)
                  (IMPLIES
                    (SAME_AS REIMBURSEMENT.SOURCE 'STATE-FUNDS)
                    (SAME_AS AUTHORIZER 'DEPARTMENT-HEAD))
                  (DIFFERENT_THAN REINBURSEMENT.SOURCE
                                 'STATE-FUNDS) )

   ATTRIBUTES: ( (FORM-FILLER NIL (RANGE . PERSON))
                 (AUTHORIZER NIL (RANGE . PERSON)) )

   PARTS: ( FILL_OUT_FORM_FIELD<1>
            FILL_OUT_FORM_FIELD<2>
            FILL_OUT_FORM_FIELD<3>
            FILL_OUT_FORM_FIELD<4>
            FILL_OUT_FORM_FIELD<5>
            FILL_OUT_FORM_FIELD<6>
            FILL_OUT_FORM_FIELD<7>
            SIGN_FORM<1>
            SIGN_FORM<2> )

   GENERALIZATIONS: ( FILL_OUT_FORM )

   ASSOCIATED-OBJECTS: ( TRAVEL-AUTHORIZATION
                         TRAVELER
                         REIMBURSEMENT ) }

     ------ KB Candidates -------

0.480  SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING
0.384  FORM
0.384  OBJECT
0.384  DEPARTMENT
```

```
0.384  UNIVERSITY
0.384  ACCOUNTING
0.384  TRAVEL-AUTHORIZATION
0.384  ISSUE_TRAVEL_AUTHORIZATION
0.384  VISIT
0.384  EVENT
0.384  TRAVELER
0.384  FLY
0.384  DRIVE
0.384  GO_SOMEWHERE
0.384  PAY
0.384  MAKE_A_RESERVATION
0.384  SOURCE
0.384  DESTINATION
0.384  TAKE_A_TRIP
0.384  TAKE_A_TRIP_AND_GET_PAID
0.288  TRAVEL
0.240  DEPARTMENT-HEAD
0.240  GRANT


      ------ Selected matches -------

GRANT FILL_OUT_FORM_FIELD DESTINATION DEPARTMENT-HEAD already in KB

No match found for FILL_OUT_TRAVEL_AUTHORIZATION

No match found for SIGN_FORM
         closest:
             0.28 0.17  TAKE_A_TRIP_AND_GET_PAID
             0.26 0.22  ISSUE_TRAVEL_AUTHORIZATION
             0.24 0.17  TAKE_A_TRIP
             0.24 0.17  VISIT
             0.22 0.14  TRAVEL

No match found for DEPARTURE-DATE

No match found for RETURN-DATE

No match found for TRUST-FUND

No match found for STATE-FUNDS
```

```
No match found for P-I
        closest:
           0.42 0.25  TRAVELER

No match found for MEANS-OF-TRANSPORT

No match found for PLANE

No match found for BUS

No match found for PRIVATE-CAR

     ------ Performed modifications -------

 MOD2613: CREATE the event Fill_out_travel_authorization

 MOD2614: ADD (Same_as form-filler 'traveler) to the Constraints
              field of Fill_out_travel_authorization

 MOD2615: ADD (Implies (same_as reimbursement.source 'state-funds)
                       (same_as authorizer 'department-head))
              to the Constraints field of
              Fill_out_travel_authorization

 MOD2616: ADD (Different_than reinbursement.source 'state-funds)
              to the Constraints field of
              Fill_out_travel_authorization

 MOD2617: ADD Authorizer to the Attribute-names field of
              Fill_out_travel_authorization

 MOD2618: ADD Form-filler to the Attribute-names field of
              Fill_out_travel_authorization

 MOD2619: ADD (Form-filler nil (range . person)) to the Attributes
              field of Fill_out_travel_authorization

 MOD2620: ADD (Authorizer nil (range . person)) to the Attributes
              field of Fill_out_travel_authorization

 MOD2621: ADD Fill_out_form_field<1> to the Parts field
```

of Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2622: ADD Fill_out_form_field<2> to the Parts field of
Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2623: ADD Fill_out_form_field<3> to the Parts field of
Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2624: ADD Fill_out_form_field<4> to the Parts field of
Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2625: ADD Fill_out_form_field<5> to the Parts field of
Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2626: ADD Fill_out_form_field<6> to the Parts field of
Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2627: ADD Fill_out_form_field<7> to the Parts field of
Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2628: ADD Sign_form<1> to the Parts field of
Fill_out_travel_authorization

MOD2629: ADD Sign_form<2> to the Parts field of
Fill_out_travel_authorization

MOD2630: ADD Fill_out_form to the Generalizations field of
Fill_out_travel_authorization

MOD2631: ADD Travel-authorization to the Associated-objects
field of Fill_out_travel_authorization
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD2632: ADD Traveler to the Associated-objects field of
Fill_out_travel_authorization

MOD2633: ADD Reimbursement to the Associated-objects field of
Fill_out_travel_authorization

MOD3476: CREATE the event Sign_form<1>

MOD3477: ADD (Equal signee 'traveler) to the Constraints field of
Sign_form<1>

MOD3478: ADD Signee to the Attribute-names field of Sign_form<1>

MOD3479: ADD (Signee 'traveler) to the Attributes field of
Sign_form<1>

MOD3480: ADD Fill_out_travel_authorization to the Part-of field of
Sign_form<1>

MOD3481: ADD Event to the Generalizations field of Sign_form<1>

MOD3482: ADD Travel-authorization to the Associated-objects field
of Sign_form<1>
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD3983: CREATE the event Sign_form<2>

MOD3984: ADD (Equal signee 'p.i.) to the Constraints field of
Sign_form<2>

MOD3985: ADD Signee to the Attribute-names field of Sign_form<2>

MOD3986: ADD (Signee 'p.i.) to the Attributes field of
Sign_form<2>

MOD3987: ADD Fill_out_travel_authorization to the Part-of field of
Sign_form<2>

MOD3988: ADD Event to the Generalizations field of Sign_form<2>

MOD3989: ADD Travel-authorization to the Associated-objects field
of Sign_form<2>
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD4496: CREATE the object Departure-date

MOD4497: ADD Travel-authorization to the Part-of field of
Departure-date
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD4498: ADD Date to the Generalizations field of Departure-date

MOD4892: CREATE the object Return-date

MOD4893: ADD Travel-authorization to the Part-of field of
Return-date
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD4894: ADD Date to the Generalizations field of Return-date

MOD5288: CREATE the object Trust-fund

MOD5289: ADD Travel-authorization to the Part-of field of
Trust-fund
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD5290: ADD Money to the Generalizations field of Trust-fund

MOD5711: CREATE the object State-funds

MOD5712: ADD Travel-authorization to the Part-of field of
State-funds
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD5713: ADD Money to the Generalizations field of State-funds

MOD6137: CREATE the object P-i

MOD6138: ADD Travel-authorization to the Part-of field of P-i
Expected: 0.192(avg) 0.192(max) 0.192(ind)

MOD6139: ADD Person to the Generalizations field of P-i

MOD6531: CREATE the object Means-of-transport

MOD6532: ADD Travel-authorization to the Part-of field of

```
        Means-of-transport
      Expected: 0.192(avg)  0.192(max)  0.192(ind)

MOD6533: ADD Plane to the Specializations field of
        Means-of-transport

MOD6534: ADD Bus to the Specializations field of
        Means-of-transport

MOD6535: ADD Private-car to the Specializations field of
        Means-of-transport

MOD6536: ADD Object to the Generalizations field of
        Means-of-transport
      Expected: 0.157(avg)  0.157(max)  0.157(ind)

MOD6993: CREATE the object Plane

MOD6994: ADD Means-of-transport to the Generalizations field of
        Plane

MOD7372: CREATE the object Bus

MOD7373: ADD Means-of-transport to the Generalizations field of
        Bus

MOD7751: CREATE the object Private-car

MOD7752: ADD Means-of-transport to the Generalizations field of
        Private-car
```

§4. FRAME-4

Clerk: Then once the traveler is back, we have a voucher form that we have to fill out - which gives a detailed account of your trip: 1) the date you left, 2) where you went, like say, you're going to take a plane, then it would be from Amherst to Bradley - there's mileage here - right now it's at 20 cents a mile. Then, let's say, you went to Washington, D.C.,

you'd list Bradley to Washington. You have to keep receipts for trains, no, sorry for planes and for the hotel. If it were to a conference, then the registration fee, and, if you drove your car own car, you have to have the odometer reading, starting and ending. Meals are a set rate. You don't need to keep receipts for that. Taxis -- they don't need receipts -- they take your word for it.

```
------ Discourse structures -------

{ OBJECT:  Private-Car

   ATTRIBUTES: ( (START-ODOMETER-READING NIL (RANGE . NUMBER))
                 (END-ODOMETER-READING NIL (RANGE . NUMBER)) )

   GENERALIZATIONS: ( OBJECT ) }

{ OBJECT:  Receipt-For-Plane

   ATTRIBUTES: ( (COST NIL (RANGE . MONEY))
                 (FLIGHT) )

   GENERALIZATIONS: ( RECEIPT )

   ASSOCIATED-EVENTS: ( EVENT-2
                        COLLECT_RECEIPTS ) }

{ OBJECT:  Receipt-For-Hotel

   ATTRIBUTES: ( (COST NIL (RANGE . MONEY))
                 (HOTEL NIL (RANGE . LODGING)) )

   GENERALIZATIONS: ( RECEIPT )

   ASSOCIATED-EVENTS: ( EVENT-2
                        COLLECT_RECEIPTS ) }

{ OBJECT:  Taxi

   ATTRIBUTES: ( (COST NIL (RANGE . MONEY)) )
```

```
   GENERALIZATIONS: ( MEANS-OF-TRANSPORT ) }

{ OBJECT:  Meal

   ATTRIBUTES: ( (COST NIL (RANGE . MONEY)) )

   GENERALIZATIONS: ( OBJECT ) }

{ EVENT:  Collect_Receipts

   GENERALIZATIONS: ( EVENT )

   ASSOCIATED-OBJECTS: ( RECEIPT-FOR-PLANE
                         RECEIPT-FOR-HOTEL ) }

{ OBJECT:  Departure-Date

   PART-OF: ( TRAVEL-VOUCHER )

   GENERALIZATIONS: ( DATE ) }

{ OBJECT:  Destination

   PART-OF: ( TRAVEL-VOUCHER )

   GENERALIZATIONS: ( LOCATION ) }

{ EVENT:  Fill_Out_Form_Field<9>

   CONSTRAINTS: ( (EQUAL FIELD 'DESTINATION) )

   ATTRIBUTES: ( (FIELD 'DESTINATION) )

   GENERALIZATIONS: ( EVENT ) }

{ EVENT:  Fill_Out_Form_Field<8>

   CONSTRAINTS: ( (EQUAL FIELD 'DEPARTURE-DATE) )

   ATTRIBUTES: ( (FIELD 'DEPARTURE-DATE) )

   GENERALIZATIONS: ( EVENT ) }
```

```
   PARTS: ( DESTINATION
            DEPARTURE-DATE )

   GENERALIZATIONS: ( FORM )

   ASSOCIATED-EVENTS: ( FILL_OUT_TRAVEL_VOUCHER ) }

{ EVENT:  Fill_Out_Travel_Voucher

   CONSTRAINTS: ( (EQUAL MILEAGE-RATE 20-CENTS-PER-MILE)
                  (SAME_AS FORM-FILLER 'TRAVELER) )

   ATTRIBUTES: ( (MILEAGE-RATE 20-CENTS-PER-MILE)
                 (FORM-FILLER NIL (RANGE . PERSON))
                 (AUTHORIZER NIL (RANGE . PERSON))
                 (DEPARTURE-DATE NIL (RANGE . DATE))
                 (DESTINATION NIL (RANGE . LOCATION)) )

   PARTS: ( FILL_OUT_FIELD<8> )

   GENERALIZATIONS: ( FILL_OUT_FORM )

   ASSOCIATED-OBJECTS: ( TRAVEL-VOUCHER
                         TRAVELER
                         REIMBURSEMENT ) }

     ------ KB Candidates -------

0.800  FILL_OUT_TRAVEL_AUTHORIZATION
0.480  PRIVATE-CAR
0.480  BUS
0.480  PLANE
0.480  MEANS-OF-TRANSPORT
0.480  P-I
0.480  STATE-FUNDS
0.480  TRUST-FUND
0.480  RETURN-DATE
0.480  DEPARTURE-DATE
0.480  SIGN_FORM<2>
```

```
0.480  SIGN_FORM<1>
0.384  DEPARTMENT-HEAD
0.384  GRANT
0.384  SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING
0.384  FORM
0.384  OBJECT
0.384  DEPARTMENT
0.384  UNIVERSITY
0.384  ACCOUNTING
0.384  TRAVEL-AUTHORIZATION
0.384  ISSUE_TRAVEL_AUTHORIZATION
0.384  VISIT

     ------ Selected matches -------

DESTINATION DEPARTURE-DATE MEAL PRIVATE-CAR
FILL_OUT_FORM_FIELD already in KB

No match found for FILL_OUT_TRAVEL_VOUCHER

No match found for TRAVEL-VOUCHER

No match found for COLLECT_RECEIPTS
        closest:
           0.24 0.20  SIGN_FORM
           0.24 0.17  VISIT

No match found for TAXI
        closest:
           0.31 0.17  PRIVATE-CAR
           0.31 0.16  BUS
           0.31 0.16  PLANE

No match found for RECEIPT-FOR-HOTEL

No match found for RECEIPT-FOR-PLANE

     ------ Performed modifications -------

 MOD8873: CREATE the event Fill_out_travel_voucher

 MOD8874: ADD (Equal mileage-rate 20-cents-per-mile) to the
```

```
          Constraints field of Fill_out_travel_voucher

MOD8875: ADD (Same_as form-filler 'traveler) to the
          Constraints field of Fill_out_travel_voucher

MOD8876: ADD Destination to the Attribute-names field of
          Fill_out_travel_voucher
      Expected: 0.192(avg)  0.192(max)  0.347(ind)

MOD8877: ADD Departure-date to the Attribute-names field of
          Fill_out_travel_voucher
      Expected: 0.192(avg)  0.192(max)  0.656(ind)

MOD8878: ADD Authorizer to the Attribute-names field of
          Fill_out_travel_voucher

MOD8879: ADD Form-filler to the Attribute-names field of
          Fill_out_travel_voucher

MOD8880: ADD Mileage-rate to the Attribute-names field of
          Fill_out_travel_voucher

MOD8881: ADD (Mileage-rate 20-cents-per-mile) to the Attributes
          field of Fill_out_travel_voucher

MOD8882: ADD (Form-filler nil (range . person)) to the Attributes
          field of Fill_out_travel_voucher

MOD8883: ADD (Authorizer nil (range . person)) to the Attributes
          field of Fill_out_travel_voucher

MOD8884: ADD (Departure-date nil (range . date)) to the Attributes
          field of Fill_out_travel_voucher

MOD8885: ADD (Destination nil (range . location)) to the
          Attributes field of Fill_out_travel_voucher

MOD8886: ADD Fill_out_field<8> to the Parts field of
          Fill_out_travel_voucher

MOD8887: ADD Fill_out_form to the Generalizations field of
          Fill_out_travel_voucher
```

MOD8888: ADD Travel-voucher to the Associated-objects field of Fill_out_travel_voucher

MOD8889: ADD Traveler to the Associated-objects field of Fill_out_travel_voucher

MOD8890: ADD Reimbursement to the Associated-objects field of Fill_out_travel_voucher

MOD9471: CREATE the object Travel-voucher

MOD9472: ADD Destination to the Parts field of Travel-voucher
Expected: 0.192(avg) 0.192(max) 0.347(ind)

MOD9473: ADD Departure-date to the Parts field of Travel-voucher
Expected: 0.192(avg) 0.192(max) 0.656(ind)

MOD9474: ADD Form to the Generalizations field of Travel-voucher
Expected: 0.157(avg) 0.157(max) 0.784(ind)

MOD9475: ADD Fill_out_travel_voucher to the Associated-events field of Travel-voucher

MOD9741: CREATE the event Collect_receipts

MOD9742: ADD Event to the Generalizations field of Collect_receipts

MOD9743: ADD Receipt-for-plane to the Associated-objects field of Collect_receipts

MOD9744: ADD Receipt-for-hotel to the Associated-objects field of Collect_receipts

MOD9993: CREATE the object Taxi

MOD9994: ADD Cost to the Attribute-names field of Taxi
Expected: 0.157(avg) 0.157(max) 0.157(ind)

MOD9995: ADD (Cost nil (range . money)) to the Attributes field of Taxi

```
MOD9996: ADD Means-of-transport to the Generalizations field of
             Taxi
         Expected: 0.188(avg)  0.192(max)  0.810(ind)

MOD10217: CREATE the object Receipt-for-hotel

MOD10218: ADD Hotel to the Attribute-names field of
              Receipt-for-hotel

MOD10219: ADD Cost to the Attribute-names field of
              Receipt-for-hotel
          Expected: 0.157(avg)  0.157(max)  0.157(ind)

MOD10220: ADD (Cost nil (range . money)) to the Attributes field
              of Receipt-for-hotel

MOD10221: ADD (Hotel nil (range . lodging)) to the Attributes
              field of Receipt-for-hotel

MOD10222: ADD Receipt to the Generalizations field of
              Receipt-for-hotel

MOD10223: ADD Collect_receipts to the Associated-events field of
              Receipt-for-hotel

MOD10534: CREATE the object Receipt-for-plane

MOD10535: ADD Flight to the Attribute-names field of
              Receipt-for-plane

MOD10536: ADD Cost to the Attribute-names field of
              Receipt-for-plane
          Expected: 0.157(avg)  0.157(max)  0.157(ind)

MOD10537: ADD (Cost nil (range . money)) to the Attributes field
              of Receipt-for-plane

MOD10538: ADD (Flight) to the Attributes field of
              Receipt-for-plane

MOD10539: ADD Receipt to the Generalizations field of
```

Receipt-for-plane

MOD10540: ADD Collect_receipts to the Associated-events field of Receipt-for-plane

§5. FRAME-5

Clerk: When you come back, you submit to whoever's doing the voucher for you, all the receipts you have, and then plan on spending a couple of minutes with that person. So she can figure your trip out. Go over the trip and its details. So she can get all she needs to know.

------ Discourse structures -------

```
{ OBJECT:  Secretary

   GENERALIZATIONS: ( PERSON ) }

{ EVENT:  Give_Receipts_To_Secretary

   ATTRIBUTES: ( DESTINATION SECRETARY )

   PART-OF: ( EVENT-2 )

   GENERALIZATIONS: ( SEND_INFORMATION ) }

{ EVENT:  Supply_Travel_Information

   ATTRIBUTES: ( DESTINATION SECRETARY )

   PART-OF: ( EVENT-2 )

   GENERALIZATIONS: ( SEND_INFORMATION ) }

{ EVENT:  Event-3

   CONSTRAINTS: ( (BEFORE TAKE_A_TRIP EVENT-2) )
```

```
   PARTS: ( TAKE_A_TRIP
            EVENT-2 )

   GENERALIZATIONS: ( EVENT )

   TEMPORAL-RELATIONSHIPS: ( (TAKE_A_TRIP BEFORE EVENT-2) ) }

{ EVENT:  Event-2

   CONSTRAINTS: ( (BEFORE GIVE_RECEIPTS_TO_SECRETARY
                          SUPPLY_TRAVEL_INFORMATION) )

   PART-OF: ( EVENT-3 )

   PARTS: ( GIVE_RECEIPTS_TO_SECRETARY
            SUPPLY_TRAVEL_INFORMATION )

   GENERALIZATIONS: ( EVENT )

   ASSOCIATED-OBJECTS: ( RECEIPT-FOR-HOTEL
                         RECEIPT-FOR-PLANE )

   TEMPORAL-RELATIONSHIPS: ( (GIVE_RECEIPTS_TO_SECRETARY BEFORE
                              SUPPLY_TRAVEL_INFORMATION) ) }

     ------ KB Candidates -------

0.800  TRAVEL-VOUCHER
0.800  FILL_OUT_TRAVEL_VOUCHER
0.512  FILL_OUT_TRAVEL_AUTHORIZATION
0.480  RECEIPT-FOR-PLANE
0.480  RECEIPT-FOR-HOTEL
0.480  TAXI
0.480  COLLECT_RECEIPTS
0.384  SIGN_FORM
0.384  PRIVATE-CAR
0.384  BUS
0.384  PLANE
0.384  MEANS-OF-TRANSPORT
0.384  P-I
0.384  STATE-FUNDS
0.384  TRUST-FUND
```

```
0.384  RETURN-DATE
0.384  DEPARTURE-DATE
0.384  GRANT
0.384  DEPARTMENT-HEAD
0.384  SEND_TRAVEL_AUTHORIZATION_TO_ACCOUNTING
0.384  FORM
0.384  OBJECT
0.384  DEPARTMENT

     ------ Selected matches -------

SECRETARY already in KB

EVENT-2 matches COLLECT_RECEIPTS (0.442)
        closest:
           0.49 0.40  COLLECT_RECEIPTS

No match found for EVENT-3
        closest:
           0.30 0.25  COLLECT_RECEIPTS
           0.23 0.17  SIGN_FORM

No match found for SUPPLY_TRAVEL_INFORMATION

No match found for GIVE_RECEIPTS_TO_SECRETARY

     ------ Performed modifications -------

 MOD11371: ADD Give_receipts_to_secretary to the Parts field of
               Collect_receipts
           Mod certainty: 0.093
           Expected: 0.261(avg)  0.480(max)  0.890(ind)

 MOD11372: ADD Supply_travel_information to the Parts field of
               Collect_receipts
           Mod certainty: 0.093
           Expected: 0.261(avg)  0.480(max)  0.890(ind)

 MOD11373: ADD Event-3 to the Parts field of Collect_receipts
           Mod certainty: 0.344
           Expected: 0.261(avg)  0.480(max)  0.890(ind)
```

```
MOD11576: CREATE the event Event-3

MOD11577: ADD (Before take_a_trip event-2) to the Constraints
               field of Event-3

MOD11578: ADD Take_a_trip to the Parts field of Event-3

MOD11579: ADD Event-2 to the Parts field of Event-3

MOD11580: ADD Event to the Generalizations field of Event-3

MOD11581: ADD (Take_a_trip before event-2) to the
               Temporal-relationships field of Event-3

MOD11849: CREATE the event Supply_travel_information

MOD11852: ADD Destination to the Attribute-names field of
               Supply_travel_information
          Expected: 0.192(avg)  0.192(max)  0.192(ind)

MOD11853: ADD Destination to the Attributes field of
               Supply_travel_information
          Expected: 0.192(avg)  0.192(max)  0.192(ind)

MOD11855: ADD Event-2 to the Part-of field of
               Supply_travel_information

MOD11856: ADD Send_information to the Generalizations field of
               Supply_travel_information

MOD12225: CREATE the event Give_receipts_to_secretary

MOD12228: ADD Destination to the Attribute-names field of
               Give_receipts_to_secretary
          Expected: 0.192(avg)  0.192(max)  0.192(ind)

MOD12229: ADD Destination to the Attributes field of
               Give_receipts_to_secretary
          Expected: 0.192(avg)  0.192(max)  0.192(ind)

MOD12231: ADD Event-2 to the Part-of field of
               Give_receipts_to_secretary
```

```
MOD12232: ADD Send_information to the Generalizations field of
              Give_receipts_to_secretary

MOD12603: ADD (Before give_receipts_to_secretary
                      supply_travel_information) to the
              Constraints field of Collect_receipts
          Mod certainty: 0.000
```

Bibliography

[AAM81] Arbib, M. A., Kfoury, A.J., and Moll, R. N.
A Basis for Theoretical Computer Science.
Springer-Verlag, New York, 1981.

[Bar85] Barto, A. G.
Learning by Statistical Cooperation of Self-Interested Neuron-like Computing Elements.
COINS Technical Report 85-11, University of Massachusetts, 1985.

[BBD86] Belkin, N. J., Brooks, N. M., and Daniels, P. J.
Knowledge elicitation using discourse analysis.
In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1986.

[BC85] Broverman, C. A. and Croft, W. B.
A knowledge-based approach to data management for intelligent user interfaces.
In *Proceedings of VLDB 11*, pages 96–104, Stockholm, 1985.

[Ben83] Bennett, J. S.
ROGET: A Knowledge-Based Consultant for Acquiring the Conceptual Structure of an Expert System.
Memo HPP-83-24, Stanford Heuristic Programming Project, 1983.

[BHR75] Bell, D., Hughes, E. R., and Rogers, J.
Area, Weight, and Volume: Monitoring and Encouraging Children's Conceptual Development.
Thomas Nelson and Sons Ltd., London, 1975.

[BW77] Bobrow, D. G. and Winograd, T.
An overview of KRL, a knowledge representation language.
Cognitive Science, 1(1), 1977.

[BW82] Bates, P. C. and Wileden, J. C.
EDL: A basis for distributed system debugging tools.
In *International Conference on Systems Science*, Hawaii, 1982.

[Car83] Carbonell, Jr., J. G.
Derivational analogy in problem solving and knowledge acquisition.
In *Proceedings of the International Machine Learning Workshop*, Oakland University, Michigan, 1983.

[CL82] Croft, W. B. and Lefkowitz, L. S.
An Office Procedure Formalism Used for an Intelligent Interface.
COINS Technical Report 82-4, University of Massachusetts, 1982.

[CL84] Croft, W. B. and Lefkowitz, L. S.
Task support in an office system.
ACM Transactions on Office Information Systems, July 1984.

[Cla84] Clancey, W. J.
NEOMYCIN: Reconfiguring a Rule-based System with Application to Teaching, pages 361–381.
Addison-Wesley, Reading, Massachusetts, 1984.

[Cla85] Clancey, W. J.
Heuristic classification.
Artificial Intelligence, 27:289–350, 1985.

[CLLH82] Croft, W. B., Lefkowitz, L. S., Lesser, V. R., and Huff, K.
POISE: An intelligent assistant for profession based systems.
In *Proceedings of the Conference on Artificial Intelligence*, Oakland University, Michigan, April 1982.

[DL82] Davis, R. and Lenat, D. B.
Knowledge-based Systems in Artificial Intelligence.
McGraw-Hill International Book Company, 1982.

[DW83] Dyer, M. and Wolf, T. C.
McDYPAR - A Demon Parser.
Technical Report, Department of Computer Science, UCLA, Los Angles, California, 1983.

[EEMT86] Eshelman, L., Ehret, D., McDermott, J., and Tan, M.
MOLE: A tenacious knowledge acquisition tool.
In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1986.

[GC86] Gruber, T., and Cohen, P.
Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition.
In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1986.

[GL80] Greiner, R. and Lenat, D.
RLL: A representation language language.
In *Proceedings of the AAAI*, pages 165–169, 1980.

[Gro78] Grosz, B. J.
Discourse Knowledge, pages 229–346.
Elsevier – North Holland, New York, 1978.

[GWP85] Ginsberg, A., Weiss, S., and Politakis, P.
SEEK2: A generalized approach to automatic knowledge base refinement.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 367–374, 1985.

[HH80] Haas, N. and Hendrix, G. G.
An approach to acquiring and applying knowledge.
In *Proceedings of the AAAI*, 1980.

[KB86] Kitto, C. M. and Boose, J. H.
Heuristics for expertise transfer: An implementation of a dialog manager for knowledge acquisition.
In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1986.

[KBJD86] Kahn, G. S., Breaux, E. H., Joseph, R. L., and DeKlerk, P.
An intelligent mixed-initiative workbench for knowledge acquisition.
In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1986.

[KM85] Kahn, G. and McDermott, J.
MUD, a drilling fluids consultant.
Technical Report, Carnegie-Mellon University, 1985.

[KNM85] Kahn, G., Nowlan, S., and McDermott, J.
MORE: An intelligent knowledge acquisition tool.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 581–584, 1985.

[Lef85] Lefkowitz, L. S.
Knowledge Acquisition for Expert Systems.
COINS Technical Report, University of Massachusetts, 1985.

[LPS86] Lenat, D. B., Prakash, M., and Shepherd, M.
CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks.
AI Magazine, 6(4):65–85, Winter 1986.

[McD83] McDonald, D. D.
Natural language generation as a computational problem: an introduction.
In *Computational Models of Discourse*, The MIT Press, Cambridge, Massachusetts, 1983.

[MMS85] Mitchell, T. M., Mahadevan, S., and Steinberg, L. I.
LEAP: A learning apprentice for VLSI design.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 573–580, 1985.

[MMW85] Marcus, S., McDermott, J., and Wang, T.
Knowledge acquisition for constructive systems.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 637–639, 1985.

[MUB82] Mitchell, T. M., Utgoff, P. E., and Banerji, R. B.
Learning by Experimentation: Acquiring and Modifying Problem-Solving Heuristics.
Technical Report LCSR-TR-31, Laboratory for Computer Science Research, Rutgers University, 1982.

[Pia64] Piaget, J.
Development and learning.
Journal of Research in Science Teaching, 2:176–186, 1964.

[Pia71] Piaget, J.
Biology and Knowledge.
The University of Chicago Press, 1971.

[Sho76] Shortliffe, E. H.
Computer-based Medical Consultations: MYCIN.
North-Holland, New York, 1976.

[Tve77] Tversky, A.
Features of similarity.
Psychological Review, 84(4):327–352, July 1977.

[vR79] van Rijsbergen, C.J.
Information Retrieval.
Butterworth & Co Ltd, second edition, 1979.

[WF83] Wright, M. and Fox, M. S.
SRL 1.5 User Manual.
Intelligent Systems Laboratory, Carnegie-Mellon University Robotics Institute, 1983.

[Wil81] Wilczynski, D.
Knowledge Acquisition in the Consul System.
Technical Report, USC/Information Sciences Institute, 1981.

[Wil86] Wilkins, D. C.
Knowledge base debugging using apprenticeship learning techniques.
In *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Alberta, Canada, 1986.

[Woo83] Woolf, B.
Tutorial Discourse Based on Context Driven Text Generation.
PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1983.

[WPML84] Woolf, B., Pustejovsky, J., McDonald, D. D., and Lander, S. *Discourse Processing in a Knowledge Acquisition Dialogue.* COINS Technical Report, University of Massachusetts, 1984.

Meta-plans That Dynamically Transform Plans

Karen E. Huff and Victor R. Lesser

COINS Technical Report 87-10
November 1987

Computer and Information Science Department
University of Massachusetts
Amherst, MA. 01003

ABSTRACT: We present a general approach to augmenting the representational power of hierarchical plan formalisms. In complex domains, this additional power is needed to capture knowledge dealing with such issues as special cases of operators and strategies for recovering when operators fail. We describe limitations inherent in operator definitions and show that they can be overcome by expressing domain knowledge as transformations on plans. These transformations *reformulate* a (partially developed) plan, tailoring it rather than contributing directly to its completion. Since transformations are operations on a world state representing the plan network, they can be formalized as meta-operators and synthesized into meta-plans. The advantage of this approach is expressive completeness as compared to introducing special-case constructs into the operator definition language.

1.0 Introduction

Planning and plan recognition systems derive their power from having general-purpose algorithms that bring domain-specific knowledge to bear; their power is directly related to the extent of the domain knowledge that can be supplied. A limiting factor on providing this knowledge is the representational adequacy of formalisms for defining domain operators. Particularly in the case of complex domains, there are problems in capturing relevant domain knowledge such as how to recover when an operator fails or when to use special case operators. The challenge is to provide this knowledge in a way that is tractable both to the planning algorithms and to the author of the domain operators.

We encountered these issues as part of designing and developing an intelligent assistant, named GRAPPLE, based on a plan recognition and planning paradigm [1,2,3,5,6]. GRAPPLE provides passive assistance by monitoring user actions, performing plan recognition to detect and correct errors in the user's plan; it provides active assistance by cooperative generation of plans to meet user-stated goals. The test domain for GRAPPLE is software development [8], specifically programming performed in a traditional language such as C. A partial library of (simplified) operators for this domain is sketched in Figure 1. These operators have the usual clauses stating goals, preconditions, effects (generally, a superset of the goals), and constraints. Primitive operators correspond to the atomic actions in the domain; complex operators have subgoals that decompose the operator into simpler steps.

1.1 Limits on Representational Power

Hierarchical plan systems, based on NOAH [12] and NONLIN [14], are particularly appropriate for handling the operator libraries of complex domains. The use of non-primitive operators allows activities to be defined at multiple levels of abstraction, with more or less detail as appropriate; this provides an orderly approach to covering all domain activities. The modularity of operator libraries facilitates several aspects of working with large numbers of operators. Following the principles of information-hiding[11], certain

```
OPERATOR build
GOAL:        status(?system,built)
PRECOND:     true
SUBGOALS:    created(?system,?baseline)
             status(?system,compiled)
             status(?system,linked)
             status(?system,tested)
EFFECTS:     SET status(?system,built)
```

```
OPERATOR release
GOAL:        status(?system,released)
PRECOND:     status(?system,built)
CONSTR:      current-release(?c)
EFFECTS:     SET status(?system,released)
             DELETE current-release(?c)
             SET current-release(?system)
             SET customer-release(?system)
```

```
OPERATOR test
GOAL:        status(?system,tested)
PRECOND:     true
SUBGOALS:    unit-tests(?system,ready)
             unit-tests(?system,run)
EFFECTS:     SET status(?system,tested)
```

```
OPERATOR link
GOAL:        status(?system,linked)
PRECOND:     status(?system,compiled)
EFFECTS:     NEW load-module ?lm
             SET status(?system,linked)
             ADD lm-for(?system,?lm)
             SET time(?system,?time)
```

```
Relationships
Build satisfies precondition of Release
Link satisfies subgoal of Build
Test satisfies subgoal of Build
```

Figure 1: A Partial Library of Simplified Software Process Operators

details can be encapsulated in one or a few operators; later, if those details must change, the effects are local, not global. Great flexibility is obtained if the decomposition of operators is always stated in terms of states to be achieved, not directly in terms of the operators that achieve those states; the operators of Figure 1 follow this style. The advantage is that when a new operator, satisfying a state required in other operators, is added to the library, the existing operators do not have to be modified to mention the new operator. This capitalizes on the fact that operators are potentially applicable in *any* context where their goals match the preconditions or subgoals of other operators. In general, operators can be written without knowledge of the other operators — either those operators that achieve the same or similar states, or those operators that require particular states in their decomposition.

In complex domains, cases arise where appropriate expressive power is lacking in the operator formalism; attempts to describe certain operators accurately can jeopardize the library modularity, or fail outright. We discuss several such problems, using examples of special cases of operators and failure recovery strategies taken from the software development domain, in the remainder of this section.

Adding special case operators to a library may require that preconditions or subgoals of existing operators be rewritten. For example, when testing a system that is intended to fix certain bugs, the programmer should run the official testcases associated with those bugs, in addition to those testcases that would otherwise be selected. One solution (that keeps testing considerations local to the set of testing operators) is to write a separate operator covering all testing needed when bugs are being fixed; its precondition restricts its applicability to systems intended to fix bugs. Now there are two operators for testing that are intended to be mutually exclusive. Therefore, the normal operator for testing must specify in its precondition that it is not applicable to cases where bugs are being fixed[1].

To accommodate other types of special cases, existing operators may have to be rewritten in artificial ways. Consider testing a system that is about to be released to a customer; such testing should include running the testcases in the regression test suite[2] (again, in additional to normally selected testcases). The precondition for this special operator concerns the existence of a goal to release the system; while the goal formula is expressible using domain predicates, the fact that a goal with this formula is currently instantiated is not expressible in domain terms. The only recourse is to write separate operators with artificially different goals. Then, operators (like *build*) that have testing subgoals will be affected, defeating the aim of encapsulating testing considerations within the testing operators. Thus, the designer of the operators must produce not only a normal *test* operator and a *test-for-release* operator, but also a normal *build* operator and a *build-for-release* operator, to ensure that the right type of testing is performed in all cases.

Expressing special cases with this brute force approach, already attended with disadvantages, breaks down entirely when multiple special cases affect a single operator; the combinatorics are intolerable from the designer's perspective. Special cases are not

[1] One could institute a fixed preference strategy to select the operator with the most specific precondition that can be satisfied. However, in general this is overly restrictive — it would prevent a car buyer from financing his purchase by selling stock to raise funds because taking out a car loan is the most narrowly applicable operator.

[2] In software engineering, regression testing is performed to ensure that bugs have not been introduced into functions that were previously shown to work correctly.

guaranteed to be simply additive with respect to the normal case. At worst, separate operators must be provided for all *combinations* of special factors.

In dealing with recovery from operator failure, there are problems both in connecting the right recovery operator with a failure situation, and in simply expressing the recovery strategy itself. Sometimes special operators are used for failure recovery, and only for failure recovery; for example, one of the actions for dealing with a compilation failure due to bugs in the compiler is to report the compiler bug. *Report-tool-bug* can be written as an operator, but how will such an operator get instantiated? Missing are the constructs indicating what goals (and therefore what operators) should be instantiated when a failure occurs. At other times, the recovery strategy may involve executing some normal operator in a special way. If the *build* operator fails because the *system* being built is faulty (as would be the case if the linker detected programming errors), then one recovery strategy is to restart the *build* process using the faulty *system* as the *baseline* from which to edit. Expressing such a strategy requires access to the variable bindings of operator instances; again, this is beyond the scope of domain predicates.

1.2 Extending Representational Power

These problems have previously been tackled separately on a case-by-case basis, introducing special operator-language constructs covering selected cases and providing domain-independent strategies that can be tailored in fixed ways. McDermott's policies [10] represent one approach to the issue of matching special case operators to the appropriate ckrcumstances. These policies derive their power from the fact that the NASL language allowed the writer to use plan-oriented constructs, such as *<policy> IMPLIES (TO-DO <task> <operator>)* or *<policy> IMPLIES (RULE-OUT <operator>)*, in addition to domain-oriented constructs. Recovery from failure of operators is addressed in SIPE [16]. There, a special error recovery language is defined; domain-independent strategies, such as reactivating a goal (RETRY in SIPE terms), are parameterized to allow deployment in operator-specific ways.

In this paper we describe a *single* formalism that extends the representational power of hierarchical planning paradigms. Our approach is based on abandoning the notion of predefining *all* operators; instead, we define transformations to be applied to *instances* of operators within a plan to create variations in response to special circumstances. Transformations are operations on some world state; in this case, the world state *is* the plan network. Therefore such transformations can be formalized as meta-operators and synthesized into meta-plans. This approach has the desired generality; it also adds to the role of meta-plans, which have been previously use to implement control strategies [13] and to capture domain-independent knowledge [15].

The primary advantage of this method is expressive generality, as compared with a collection of *ad hoc* operator language extensions. Any aspect of an operator definition (such as preconditions, subgoals, constraints, or effects), as well as any aspect of an operator instance (such as bindings of variables or ancestor operator instances) can be accessed or modified. The transformational approach also addresses some practical problems associated with developing and maintaining a complete library of operators. Because knowledge of exceptions is partitioned from knowledge of normal cases, the two issues can be tackled separately. The process of writing operators is further improved because multiple transformations can apply to a single operator, thereby preventing combinatorial explosion in numbers of operators.

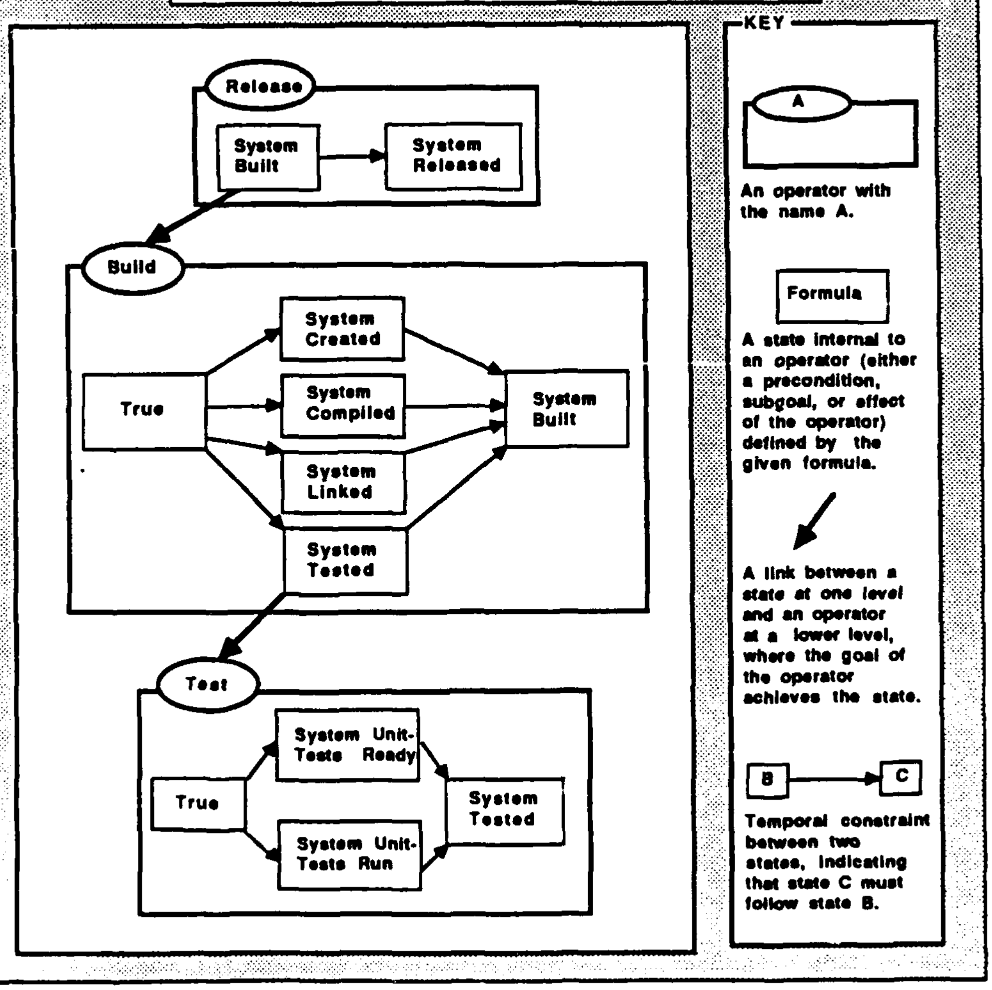
In the remainder of the paper, we introduce the transformational approach with some specific examples from the software development domain. Then, we discuss how the transformations are formalized and expressed as meta-operators; both the state description and required operator constructs are covered. Finally, we present status and conclusions.

2.0 Transformations on Plans

2.1 Plans As Networks

The basic data structure of a planner or plan recognizer is a hierarchical plan network as first developed in NOAH[12]. An example of such a network (using some of the plans of Figure 1) is given in Figure 2. There, a vertical slice through a network covering three hierarchical levels is shown, with the highest level at the top of the figure. Downward arrows between levels connect desired states with operators chosen to achieve them. Such

FIGURE 2: Example Hierarchical Plan Network

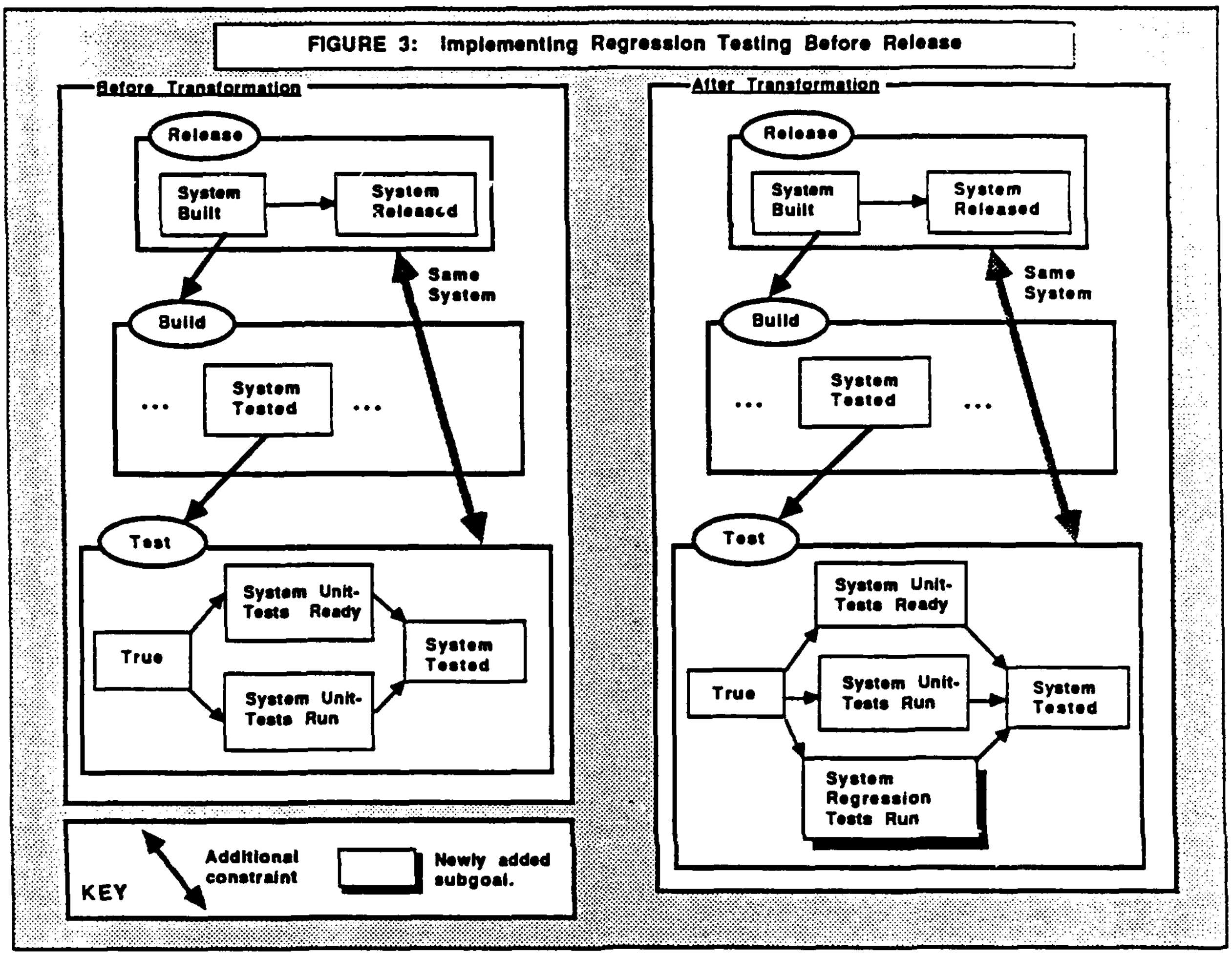


instantiated operators consist of additional states describing the internals of the operator: preconditions, subgoals, and effects. Arrows within levels show how the achievement of certain states is partially ordered with respect to time (some orderings are specified by the operator definitions and other orderings are imposed to resolve interactions). Orderings are propagated from level to level, but have been omitted to simplify the figure.

Both planning and plan recognition involve building a complete plan network. This is done by actions such as choosing operators to achieve states, instantiating these operators, and resolving conflicts between newly revealed states and existing states. Each such action can be thought of as taking the plan network one step closer to completeness. In contrast, a transformation will reformulate the current state of the network, with the effect of changing the solution set that will be pursued to complete the network. Such a reformulation is necessary either because the existing state of the network does not accurately reflect special circumstances or because other actions have reached a dead end (for example, a plan failure has occurred). Reformulating the network represents a permanently- instantiated objective of the planner/recognizer. Thus, the execution of (top-level) transformations will be data-driven: they will be applied whenever the current state of the network indicates an opportunity to do so.

2.2 Example: A Special Case

Software development is an example of a domain where a large part of the knowledge about how actions are carried out is concerned with special cases. Consider a transformation that implements the requirement to do regression testing before a release. When expressed precisely, the transformation affects an instance of *test* occurring as part of the expansion of *release*. To be entirely safe, one additional restriction should be given: that the *system* being tested is the same as the *system* being released. This will allow other testing instances to occur in the same expansion (such as running a testcase to help decide what editing changes are needed), while ensuring that regression testing is required on the right one. Expressing this condition requires access to the dynamic correspondence between the variable names used in the two operators. The BEFORE case of Figure 3 shows one situation in which this transformation is applicable.



Assuming the *test* operator of Figure 1, the effect of the transformation is to add an additional subgoal to run the regression test cases. The formula defining the new subgoal is supplied explicitly in the transformation -- it need not have appeared previously in the plan network. Only the one operator instance is modified; the basic operator definition for *test* is unchanged. The results of applying this transformation are shown as the AFTER case in Figure 3.

2.3 Example: Failure Recovery

Software development is also a domain where there are many causes of failure, including system problems, tool problems and programmer error. In particular, given that much work is carried out on a trial-and-error basis, failures due to programmer error are to be

expected frequently. Consider the case of the *link* operator failing to produce a load module because errors made by the programmer were detected. In fact, decisions about recovery from this failure are not made at the local level of the *link* operator; a *link* operator failure implies that the parent operator has failed, and the appropriate recovery strategy is dictated by what that parent operator is.

The parent operator will usually be the *build* operator. If the *build* operator has failed and the *system* that was built is faulty, one recovery scenario is to go through the whole *build* process again; but, instead of starting from the same *baseline* used in the original *build* instance, this new *build* instance will start with the faulty *system* as the *baseline*. That is, the new *build* instantiation will use as the binding of its *baseline* variable the binding of the *system* variable from the failed *build* instantiation.

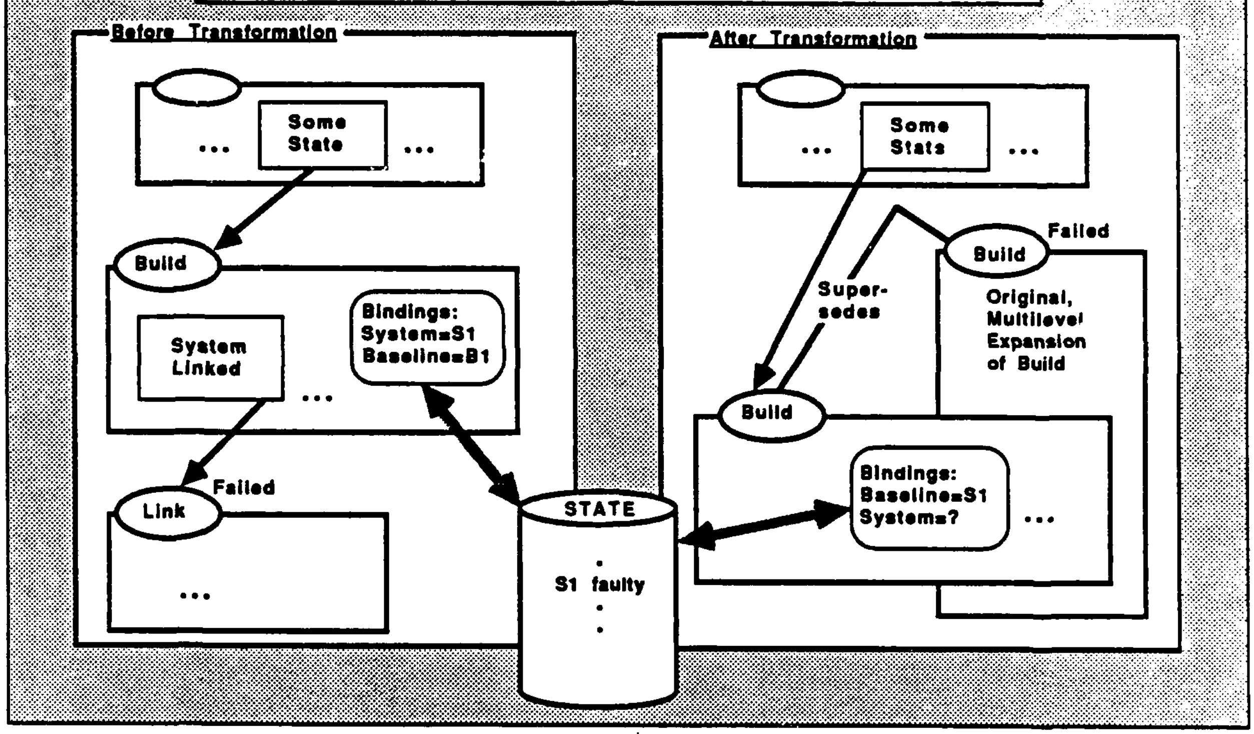
These strategies can be expressed in two separate transformations. The first transformation applies to instances of *link* that have failed; its effect is simply to mark the parent operator instance as failed. The second transformation applies to instances of *build* that have failed; its effect is to create a new instantiation of the *build* operator, and to fix the binding of the *baseline* variable in that instantiation to be equal to the binding of the *system* variable from the "superseded" instantiation of *build*. This is shown in Figure 4.

2.4 Other Examples

The software development domain is particularly rich in examples that demonstrate the generality of the transformational approach. Some additional generic uses of transformations, beyond representing special cases and straightforward failure recovery strategies, are these:

- Transformations can be used to maintain desirable domain states in a flexible manner (McDermott used policies this way[10]). That is, whenever an undesirable state obtains, a goal can be posted to reestablish the desired state. This is more forgiving than preventing the undesired state absolutely. As an example, programmers generally follow a set of rules about how files are allocated to directories. However, in the heat of activity, a file may be created

Figure 4: Recovery From Failure of Link and Build



in the "wrong" directory. A transformation could trigger on this and instantiate a goal to move the file to the proper directory.

- One method of handling an adverse interaction between operators for achieving parallel goals is to place temporal constraints on the order in which the operators are executed. This method has the advantage of being domain-independent, but there may be domain-dependent techniques as well. These can be captured in transformations whose preconditions are that adverse interactions between two planned actions have been detected. In the software domain, the operator that copies the contents of one file into another can be inserted into a plan to correct some types of interactions. However, some explicit clue, such as a transformation, is needed to ensure that copy is considered for handling interactions.

- Transformations can be used to apply shortcuts in just those situations where the shortcut is safe. A shortcut amounts to substituting one goal which is "easier" to achieve for another which is "harder" to achieve. The safety of the shortcut may involve the context in which the goal is instantiated, so the meta-level constructs of the transformation are doubly necessary. In the software domain, if editing a source module consists of cosmetic changes only, then an alternative to recompilation is simply to acquire (and place in the appropriate

directory) the object module of the previous version (assuming no include modules were also changed). However, it is bad practice to do this on a release to a customer. Only by expressing this in a transformation can we ensure that good practice is followed.

- In some cases when operators fail, the recovery strategy may involve rephrasing the goal in order to proceed with the overall plan. In these cases, a failure indicates that the goal itself (in all its detail) cannot be accomplished; but there may be a related goal that will suffice. A transformation, whose precondition is that an operator failed to achieve its goal and whose effects are to substitute a different goal for the original goal, can express this strategy. A software example arises if the compiler blows up when directed to compile at its highest optimization level. A well-established strategy is to try again with optimization turned off. If this results in a successful compilation, the programmer will settle for a load module which is only partially optimized.

- Transformations can apply to a specific operator (such as the *test* operator examples given earlier), or to any operator having a particular characteristic, such as a specific goal or subgoal formula. They can also apply to arbitrary characteristics of operator definitions, such as any operator having a particular type of parameter or parameter binding. In a multi-user system, when the number of users logged-in is below a certain threshold, then commands will be submitted for foreground execution rather than to a background queue. Suppose all operators representing CPU-intensive commands are written with an explicit parameter set for background execution. Then, one transformation, applying to any operator utilizing the background queue, can handle foreground/background selection. This transformation saves the author of operators from writing an additional version of each CPU-intensive operator.

- The examples of transformations given so far are relatively simple, most often requiring one operation on the plan net. However, transformations can be arbitrarily complex. In the software domain, recovering from failed operators includes deleting extraneous files. One transformation could identify certain files created by operators in the expansion of a failed operator and instantiate goals to delete them. This transformation applies one change (instantiate a goal with specific variable bindings) many times (for each selected file).

- Another complex transformation in the software domain applies to the conservative editing style of frequently saving a snapshot of the file being edited; here the intermediate snapshots must eventually be deleted. This is a complex transformation involving two separate (but related) changes in the plan net: one change instantiates goals to save snapshots, and the other change instantiates goals to delete all but the desired version.

- A final use of transformations is to allow sharing of a generic operator library among several applications, where each application has special requirements. In the software domain, several projects can share a generic operator library, if each expresses project-specific policies as transformations. Then, one project can require that a particular analysis tool be run before a customer release, without affecting whether other projects use the same tool in the same way.

3.0 Formalizing the Transformations

Because transformations are operations on the plan network, they can be formally represented as meta-plans, synthesized from meta-operators. Meta-plans are not a new idea. Procedurally-implemented meta-plans were introduced by Stefik[13] to pursue control issues in planning. Declarative meta-plans were defined by Wilensky[15] in order to share knowledge between a planner and a plan recognizer. Neither of these meta-plan systems was used to modify operator instances by adding new subgoals, changing constraints and so forth. Meta-plans that could modify steps or change parameter bindings were defined for a natural language dialog understanding system[9]. In these meta-plans, the modifications were meta-plan parameters which were bound from information in the utterances.

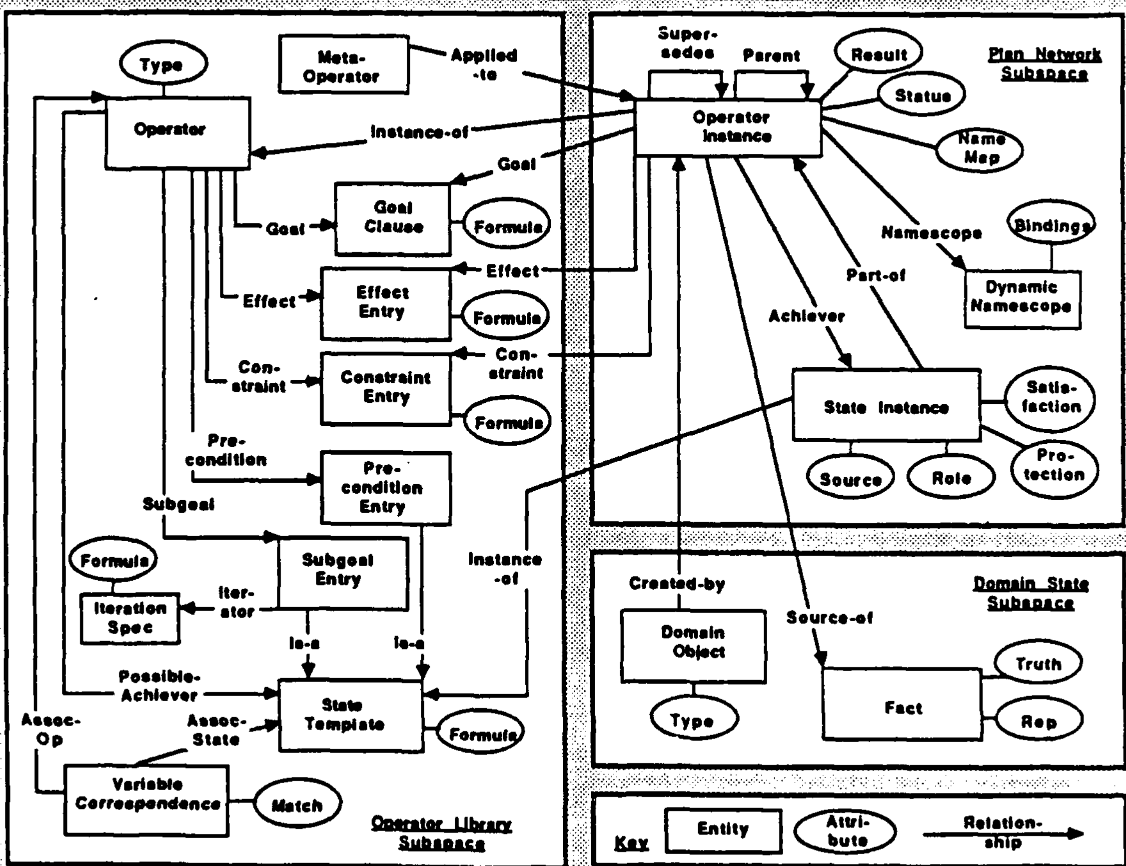
3.1 The Meta-level State Schema

The meta-level state schema covers most of the internal data structures used in planning. The entity-relationship (ER) model of data [4] provides a useful way to visualize such a complex state schema. In the ER model, there are entities which have attributes and participate in relationships with other entities. There is a straightforward translation between the ER model and predicate calculus, whereby relationships and attributes correspond to predicates. Semantic constraints can be expressed as axioms in predicate calculus. Other axioms can be used to define additional predicates and to define appropriate functions.

The ER diagram of the meta-level state schema is given in Figure 5. The objects and relationships shown are representative, not exhaustive. The schema describes operators as they are defined in the GRAPPLE formalism, and addresses the requirements of plan recognition (the context in which we are currently implementing the meta-plans).

The state schema for the meta-plans contains objects and predicates organized into three subspaces: operator library, plan network, and the domain state. The operator library subspace describes all components (preconditions, effects, etc.) of all operators, and their formulas and (static) variable names. The plan network subspace describes the hierarchy of operator instances: their dynamic status (started, completed, failed...), their internal

Figure 5: Meta-Level State Schema



states and status (pending, achieved, protected...). Also associated with operator instances are the dynamic name scopes and variable binding information needed to evaluate operator formulas. The domain state represents the truth value of all domain predicates. Additional predicates cross subspace boundaries, relating operator instances back to their definitions, and operator instances to the domain predicates they affected.

The state schema is designed so that it represents a single choice from among the competing interpretations of a series of actions. Thus, if an operator can achieve the subgoals of two other operators, this will be represented using two states, each in a separate context. During recognition, there will usually be multiple contexts that are active; an important component of the recognizer comprises strategies for focusing among these alternatives, as well as for selecting the operations applied to an alternative. Thus we make the same distinction as in [13] between operations in planning space and the control strategies by which those operations are selected. The transformational meta-operators add to the number of operators subject to control decisions.

3.2 Meta-operator Constructs

Because the transformations are complex, expressing them as operators requires a language engineered for "real-world" use. Clearly, effects of operators must be able to create new objects (for example, a new subgoal instance). Some transformations (such as the one to delete extraneous files) require a facility for iterating subgoals: that is, for repeatedly achieving a subgoal formula over a set of bindings. Conveniently, all the needed facilities were already available in the formalism we designed to handle the complex domains anticipated for GRAPPLE [7]. No new facilities (except a keyword to distinguish between operators and meta-operators) were needed.

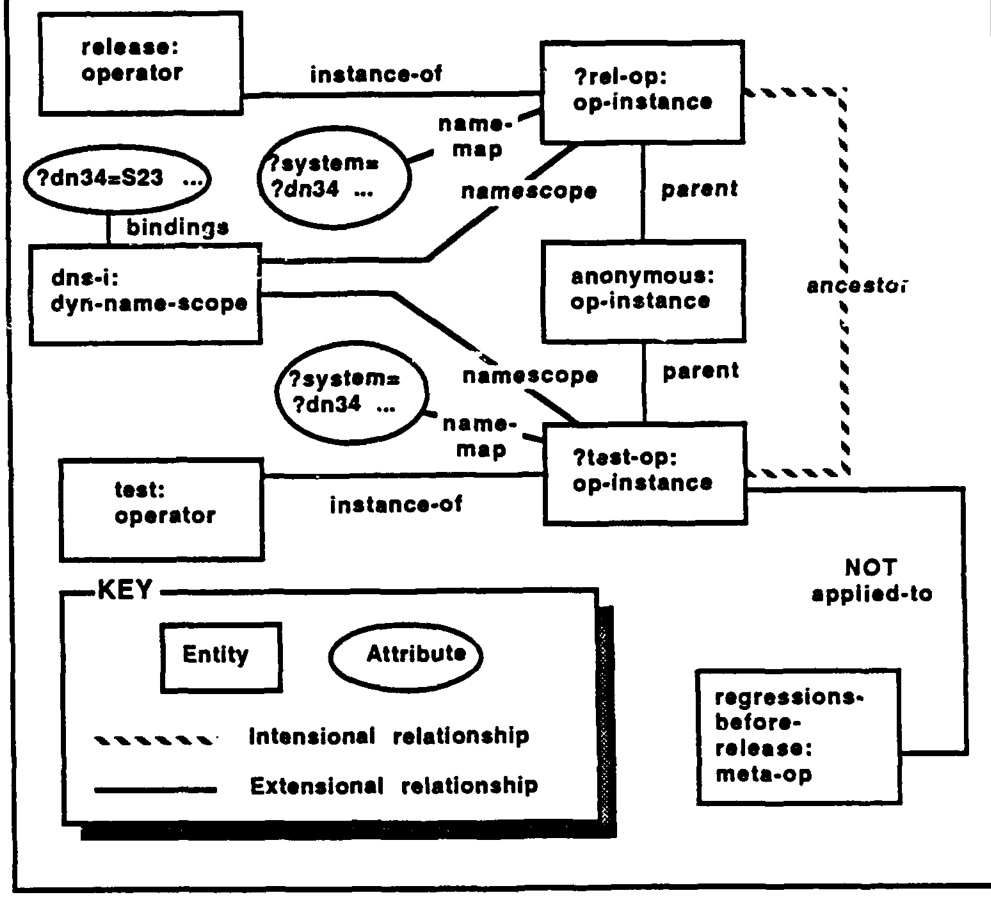
The transformation for regression testing before release is shown as a meta-operator (expressed in GRAPPLE notation) in Figure 6. This will be a top-level operator assuming that its goal does not match the precondition or subgoal of other meta-operators; therefore, it will be executed when its precondition becomes true. That will happen when there is an instance of *test* whose ancestor is an instance of *release* AND when the *system* variables of

Figure 6: Meta-operator for Regression Testing

```
(METAOPERATOR   regressions-before-release

(GOAL           applied-to(regressions-before-release,?test-op))

(PRECOND        (STATIC instance-of(?test-op,test) AND
                ancestor(?test-op,?rel-op) AND
                instance-of(?rel-op,release) AND
                same-dynamic-name(system,?rel-op,
                  system,?test-op) ) AND
                NOT applied-to(regressions-before-release,
                  ?test-op) ) )

(EFFECTS        (NEW state-instance ?regressions)
                (NEW subgoal-entry  ?regr-subgoal)
                (NEW iteration-spec ?iterate-info)
                (ADD part-of(?regressions, ?test-op))
                (ADD instance-of(?regressions, ?regr-subgoal))
                (ADD iterator(?regr-subgoal, ?iterate-info))
                (ADD source(?regressions, metaplan))
                (ADD role(?regressions, subgoal))
                (ADD protection(?regressions, not-protected))
                (ADD satisfaction(?regressions, unknown))
                (ADD formula(?regr-subgoal,
                      tested-on(?system,?regr-case)) )
                (ADD formula(?iterate-info,
                      in-regression-suite(?regr-case)) )
                (ADD applied-to(regressions-before-release,
                    ?test-op) ) ) )
```

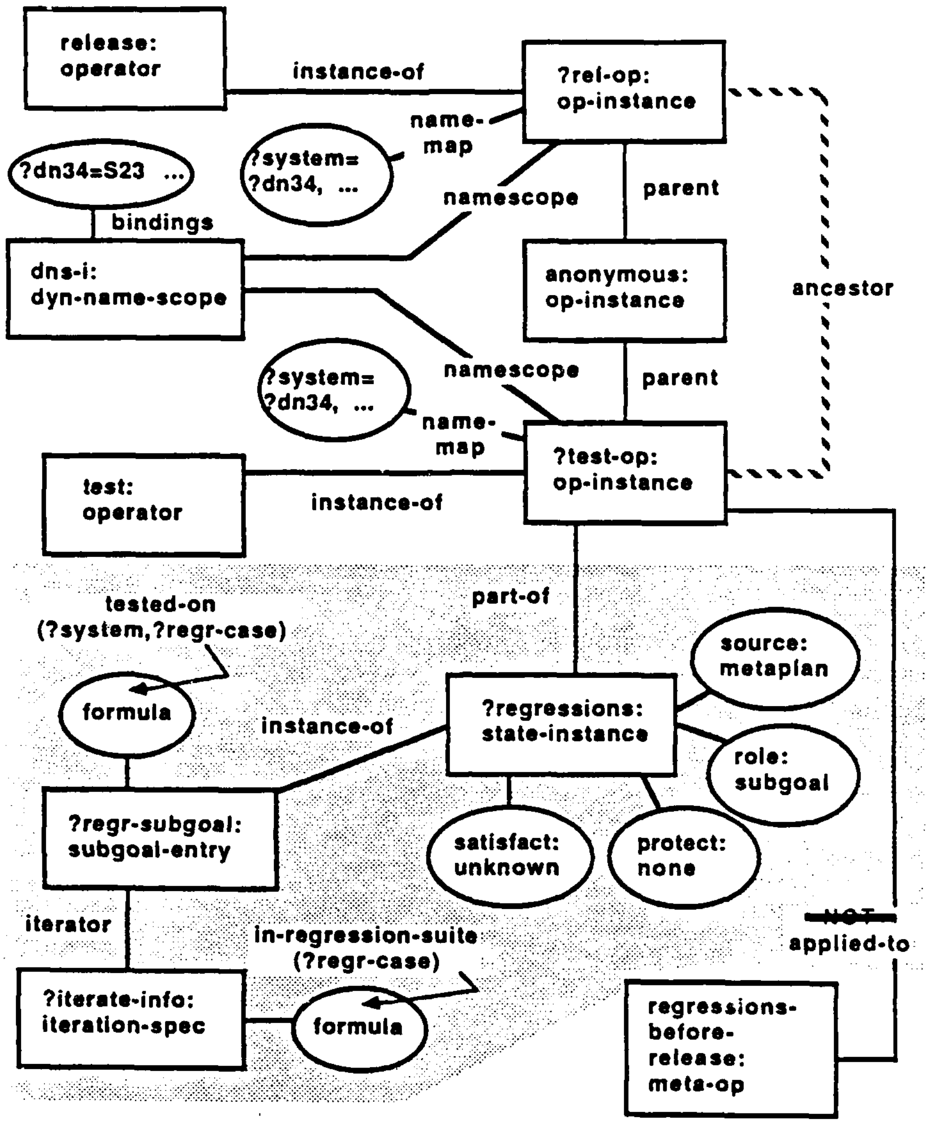
both operators correspond (e.g., are mapped to the same dynamic name). The precondition carries the keyword STATIC, meaning that no explicit attempt should be made to render it true. A state in which the transformation precondition is true is diagrammed in Figure 7.

Figure 7: Example Meta-level State Satisfying Precondition of Transformation



Performing the transformation is simply a matter of realizing the effects of the meta-operator in this case, because there are no subgoals to be achieved. These effects are all directed at creating a new state instance as part of the *test* operator instance. The new state instance has a supporting subgoal entry that defines the state formula and, since it is an iterated subgoal, the iteration formula. Note that the meta-operator contains these formulas explicitly; they consist of domain predicates and variable names in the static name scope of the *test* operator definition. After the transformation has been executed, the precondition will no longer be true for this instance of *test*; thus, this transformation is not meant to be applied more than once to the same situation. The state shown in Figure 7 becomes the state diagrammed in Figure 8 after the transformation is applied; changes are highlighted.

Figure 8: Meta-level State after Transformation



The transformation exporting the failure of the *link* operator to its parent operator is shown in Figure 9, to show how a goal for failure recovery is expressed.

Figure 9: Meta-operator for Link Failure

(METAOPERATOR	propagate-link-failure
(GOAL	status(?link-op,failure-processed))
(PRECOND	(STATIC status(?link-op, failed) AND instance-of(?link-op,link) AND query(?link-op, *faulty(?system)*)))
(CONSTRAINTS	(parent(?link-op,?parent-op))
(EFFECTS	(DELETE status(?parent-op,in-progress)) (DELETE status (?link-op,failed)) (ADD status(?parent-op,failed)) (ADD status(?link-op,failure-processed))))

4.0 Status and Conclusion

Transformational meta-plans provide a powerful means of capturing and applying additional domain knowledge, which is key to achieving more powerful planning systems. Defining domain knowledge about special cases and failure recovery via meta-operators provides an expressively complete approach, which obviates the need for special-purpose language extensions. This approach also expands the role that meta-planning plays in a planning architecture. The resulting transformations represent a special class of operations on plan networks: operations that reformulate a network rather than solve it. As an additional benefit, this approach eases the writer's task of providing thorough coverage of domain actions. We are currently implementing the transformations as part of the GRAPPLE plan recognizer. This implementation uses Knowledge Craft™ and capitalizes on its facilities for context management, object schema, and integrated Prolog features. We are continuing to explore the role of deeper domain knowledge in planning systems, with particular emphasis on its relationship to control.

Knowledge Craft™ is a trademark of Carnegie Group Incorporated.

5.0 References

[1] Broverman, C.A., and W.B. Croft, "A Knowledge-based Approach to Data Management for Intelligent User Interfaces", *Proceedings of Conference for Very Large Databases*, 1985.

[2] Broverman, C.A., K.E. Huff, and V.R. Lesser, "The Role of Plan Recognition in Intelligent Interface Design, *Proceedings of Conference on Systems, Man and Cybernetics*, IEEE, 1986, pp. 863-868.

[3] Carver, N., V.R. Lesser and D. McCue, "Focusing in Plan Recognition", *Proceedings of AAAI*, 1984, pp. 42-48.

[4] Chen, P.P., "The Entity-relationship model: Toward A Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, no. 1, March 1976, pp. 9-36.

[5] Croft, W.B., L. Lefkowitz, V.R. Lesser and K.E. Huff, "POISE: An Intelligent Interface for Profession-based Systems", Conference on Artificial Intelligence, Oakland, Michigan, 1983.

[6] Croft, W.B., and L.S. Lefkowitz, "Task Support in an Office System", *ACM Transactions on Office Information Systems*, vol. 2, 1984, pp. 197-212.

[7] Huff, K.E. and V.R. Lesser, "The GRAPPLE Plan Formalism", Technical Report 87-08, Department of Computer and Information Sciences, University of Massachusetts, Amherst, 1987.

[8] Huff, K.E. and V.R. Lesser, "A Plan-based Intelligent Assistant That Supports the Process of Programming", Technical Report 87-09, Department of Computer and Information Sciences, University of Massachusetts, Amherst, 1987.

[9] Litman, D., "Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues", PhD. Dissertation (also TR 170), Department of Computer Science, University of Rochester, 1985.

[10] McDermott, D., "Planning and Acting", *Cognitive Science*, vol. 2, 1978, pp. 71-109.

[11] Parnas, D., "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM*, vol. 15, no. 3, March 1972.

[12] Sacerdoti, E.D., *A Structure for Plans and Behavior*, Elsevier-North Holland, New York, 1977.

[13] Stefik, M., "Planning and Meta-planning", *Artificial Intelligence*, vol. 16, 1981, pp. 141-169.

[14] Tate, A., "Project Planning Using a Hierarchical Non-linear Planner", Dept. of Artificial Intelligence Report 25, Edinburgh University, 1976.

[15] Wilensky, R., "Meta-Planning: Representation and Using Knowledge About Planning in Problem Solving and Natural Language Understanding", *Cognitive Science*, vol. 5, 1981, pp. 197-233.

[16] Wilkins, D.E., "Recovering From Execution Errors in SIPE", TN 346, SRI International, January, 1985.

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

Multiple Knowledge Sources in Intelligent Teaching Systems

Beverly Woolf, University of Massachusetts
Patricia A. Cunningham, The Hartford Graduate Center

Intelligent teaching systems are emerging as a possible solution to the nation's large training problem in government, academy, and industry. But since the few systems that have been built were customized for specific applications, few guidelines or tools exist for building intelligent teaching systems. As a result, building such systems remains a black art—a pretechnology requiring considerable experimentation and effort while producing minimal results. In addition, existing systems show intelligence within a narrow specialization. Like other expert systems, tutoring systems display varying levels of expertise.[1] In every case, an intelligent tutor has less breadth of scope and flexibility than does its human counterpart. Incorporating community expertise could increase the limited competence of current tutors.

Model: *Mother Theresa, 1981.*

We believe that intelligent tutoring systems can be designed using a community memory, with multiple experts contributing their teaching and learning knowledge. The concept of knowledge base as community memory reflects the fact that knowledge is often distributed, incomplete, and acquired incrementally—especially in tutoring systems where the domain expert, cognitive scientist, and teaching expert are typically not the same person.[1] Experience with commercially successful expert systems such as R1[2] and the Dipmeter Advisor[3] suggests that using knowledge from a single expert can produce systems foreign to other system users—systems having conceptual holes. In the case of the Dipmeter Advisor, the expert model solved problems in an uncommon way, creating blind spots in the knowledge base.[1] To develop a community memory for tutoring systems, we need to create a framework within which

An earlier version of this article appears in the *Proceedings of the American Association of Artificial Intelligence*, Seattle, Wash., July 1987.

teaching experience and domain wisdom can be saved. We must be able to (1) modify this framework, and (2) augment it as time passes.

We will study the development of three intelligent tutors, describing how lessons about acquiring knowledge from multiple experts were applied toward building each system. In particular, we will look at tools and methodologies for knowledge acquisition that might be transferred to other systems. We will first describe the role of computers in education, and then examine our three case studies in detail. Finally, we will extrapolate from these systems possible tools and criteria for building intelligent tutors.

Computers in education

Education is in trouble. Tough classroom problems, lack of public support, and inappropriate policies have left educators engaged in an uphill battle to reverse deficiencies in student learning and teacher training. This dilemma has fostered sometimes unrealistic expectations on the state of intelligent tutoring systems; hardware and software potential have led to exaggerated stories about the possible achievements of an as yet undeveloped technology.

As Figure 1 illustrates, innovation in computer technologies can be viewed as an S-curve measuring the relationship between the effort put into improving a product or process and the results one gets back from that investment.[4] Preintelligent tutoring systems, or computer-aided instructional systems, are at the end of their S-curve (Figure 1a); consequently, increased effort brings little additional performance. Expert systems (Figure 1b) are beginning to emerge from the low part of their S-curve as uniform and easy-to-use technologies like AI shells are developed, speeding production and performance. Intelligent tutoring systems (Figure 1c) are at the beginning of their S-curve; thus, they require substantial effort and new tools before they can produce substantial results.

Preintelligent tutoring systems achieved high performance in many areas by encoding built-in commitments regarding how their knowledge would be used.[5,6] These systems represented knowledge implicitly within the specific command that determined (and thus predefined) system input/output. As a result, each system required excessive development time.

The first hour of computer-aided instruction typically required about 200 hours of programmer preparation—an amount possibly exceeded by the preparation time required for building intelligent teaching systems. However, each additional hour of instruction for preintelligent systems also required about 200 hours of programmer preparation. Domain or teaching knowledge must be implicitly defined in the very question or explanation for which the knowledge is intended. This led to systems that could not automatically query students about concepts that had been previously defined.

In comparison, intelligent tutors are designed to make their knowledge explicit and uncommitted, thereby utilizing a single piece of knowledge in many ways. For example, by explicitly representing a concept such as force or acceleration, a physics tutor should have some flexibility in the use of that knowledge—and should be able to test students about the concept, describe the concept, demonstrate it within a simulation, and answer questions about it.

Case studies

The following three example tutors all required multiple sources of knowledge expressing their knowledge explicitly. From these case studies, we will attempt to derive tools and methodologies that can be used to acquire knowledge for the next intelligent tutor generation.

RBT for teaching complex industrial processes. The first tutor is fully implemented, tested, and now being used for training in nearly 60 industrial sites across America. It is still being augmented based on formal evaluations of student response to the system. RBT (the recovery boiler tutor) was built for a kraft recovery boiler—the type of boiler found in paper mills throughout the United States.[7]

Built by the J.H. Jansen Company and sponsored by the American Paper Institute, RBT provides multiple explanations and tutoring facilities tempered to the individual user (a control room operator). The tutor is based on a mathematically accurate formulation of the boiler and provides an interactive simulation complete with help, hints, explanations, and tutoring (see Figure 2).

Students can initiate any of 20 training situations, emergencies, and operating conditions—or they can ask that an emergency be chosen for them. Students can also accidentally trigger an emergency as a result of their actions on the boiler. Once an emergency has been initiated, students are encouraged to adjust

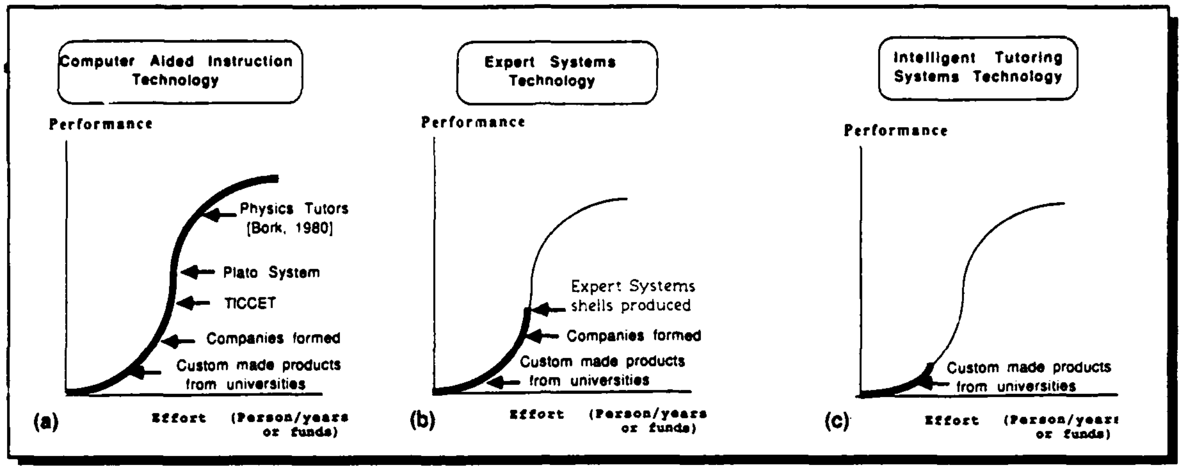


Figure 1. The S-curve of three computer technologies: **(a)** computer-aided instruction technology; **(b)** expert systems technology; **(c)** intelligent tutoring systems technology.

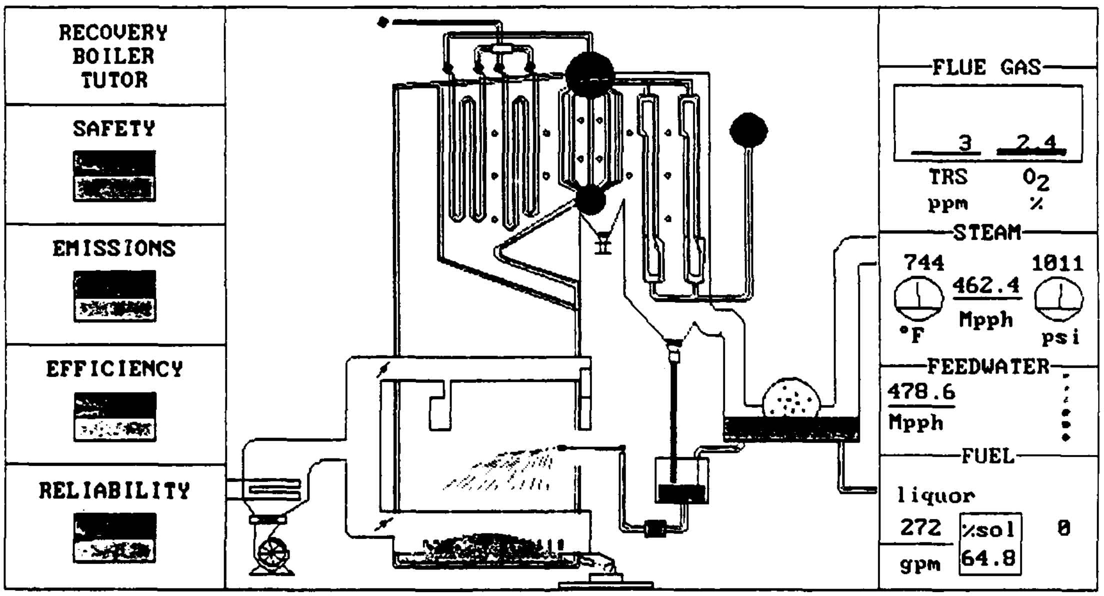


Figure 2. A sectional view of the recovery boiler.

meters and perform actions on the simulated boiler to solve the emergency. The system challenges student operators to solve new problems while the system monitors and advises those operators. RBT can recognize optimal, less than optimal, and clearly irrelevant actions. Operators can continue their freewheeling or purposeful problem-solving behavior while the tutor offers help, hints, and explanations—plus providing tutoring advice when needed or requested. Operators are expected to observe the impact of their actions on the simulated boiler and to react before the tutor advises them regarding potential problems.

An example dialogue. In the example interaction below between student and tutor, a partial blackout is in progress that will lead to combustion process deterioration if no action is taken. As students change set-point controllers and request information about the boiler, the tutor selectively discusses the optimality of their actions and suggests how they might better focus their actions or utilize their data. An important feature to note about this dialogue is that at any point during the simulated emergency there are many actions that operators might take and, as the problem worsens, an increasing number of actions that they *should* take to correct the operating conditions. Thus, an immediate and correct response might require only one action, such as rodding the primary air ports. But a delayed response causes the situation to worsen, requiring the addition of auxiliary fuel.

Meters indicate that CO, SO2, and O2 readings are abnormally high. An alarm buzzer sounds, beginning the following dialogue:

> **Tutor:** What action are you going to take?
> **Student:** Check smelt spout cooling water. [The student's action does not result in a change in the boiler.]
> **Tutor:** It looks as though the liquor isn't burning . . .
> **Student:** Help. What is going on? [The student requests help.]
> **Tutor:** A partial blackout is occurring and this causes TRS, CO, SO2, and O2 to increase. Steam flow and steam temperature are dropping. I think part of the bed is not burning.
> **Student:** Rod the primary air ports. [A small improvement in the boiler readings can be observed.]
> **Tutor:** Your action is correct; however, at this time rodding alone is not effective . . .
> **Student:** Raise the liquor temperature. [The student's action does not reverse the emergency and a new alarm buzzer sounds.]
> **Tutor:** Rodding the air port apparently was not enough to undo the blackout.
> **Student:** Increase the oil fuel supply. [The student's action reverses the emergency.]
> **Tutor:** That solved the problem alright. Good thinking.
> Analysis of the problem: You had a partial blackout caused by plugged primary air ports and a cold bed. Partial blackouts can be effectively treated through a combination of rodding the primary air ports and adding more heat. The problem can be avoided by keeping the air ports clean.

This dialogue was not actually produced in natural language; student input was handled through menus and tutor output produced by cutting text from emergency-specific text files loaded when the emergency was invoked. Operator interactions are handled through a hierarchy of menus enabling such activities as checking for a tube leak or rodding the smelt spout as well as selecting the alarm board or control panel board.

While the simulation of the recovery boiler is running, operators can view the boiler from many directions and can focus on several components; for example, the fire bed in Figure 3. The tutor provides assistance through visual clues such as a darkened smelt bed, acoustic clues, ringing alarm buzzers, textual help, explanations, and dialogues. Operators can request up to 30 process parameters on the complete panel board, view an alarm board (not shown), change 20 set points, and ask menu questions such as "What is the problem?" "How do I get out of it?" "What caused it?" and "What can I do to prevent it?" These questions are answered by cutting text from a file which was loaded with the specific emergency. These questions do not provide the basis of the tutor's knowledge representation, which is described elsewhere.[7] Operators can request meter readings, physical and chemical reports, and dynamic trends of variables (see Figure 4). All variables are updated in real time every 1 or 2 seconds.

In addition to providing information about the explicit variables in the boiler, RBT provides reasoning tools designed to aid students in reasoning about implicit processes in the boiler. One such tool is composite meters (shown on the left sides of Figures 2 through 5) recording the state of the boiler using synthetic measures for *safety, emissions, efficiency,* and *reliability* of the boiler. The meter readings are calculated from complex mathematical formulas that would rarely (if ever) be used by operators to evaluate the boiler.

For instance, the safety meter is a composition of seven independent parameters including steam pressure, steam flow, steam temperature, feed-water flow, drum-water level, firing-liquor solids, and combustibles in the flue gas. Meter readings allow students to make inferences about the effect of their actions on the boiler using characteristics of the running boiler. These meters are not presently available on existing pulp and paper mill control panels; however, if they prove effective as training aids, they could be incorporated into actual control panels.

Other reasoning tools include trend analyses (see Figure 5) and animated graphics such as shown in the Figures above. Animated graphics provide realistic and dynamic drawings of the several boiler components such as steam, fire, smoke, black liquor, and fuel. Trend analyses show how essential process variables interact in real time by allowing operators to select up to 10 variables including liquor flow, oil flow, and air flow—and to plot each against the others and against time.

Each student action, be it a set-point adjustment or a proposed solution, is recorded in an accumulated response value reflecting overall operator scores and how successful (or unsuccessful) operator actions have been and whether actions were performed in sequence with other relevant or irrelevant actions. This accumulated value is not presently used by the tutor, but the notation might be used to sensitize the tutor's future responses to student records. For instance, if operators have successfully solved a number of boiler emergencies, the accumulated value might be used to temper subsequent tutoring so that it is less intrusive. Similarly, if student performance has been poor, the

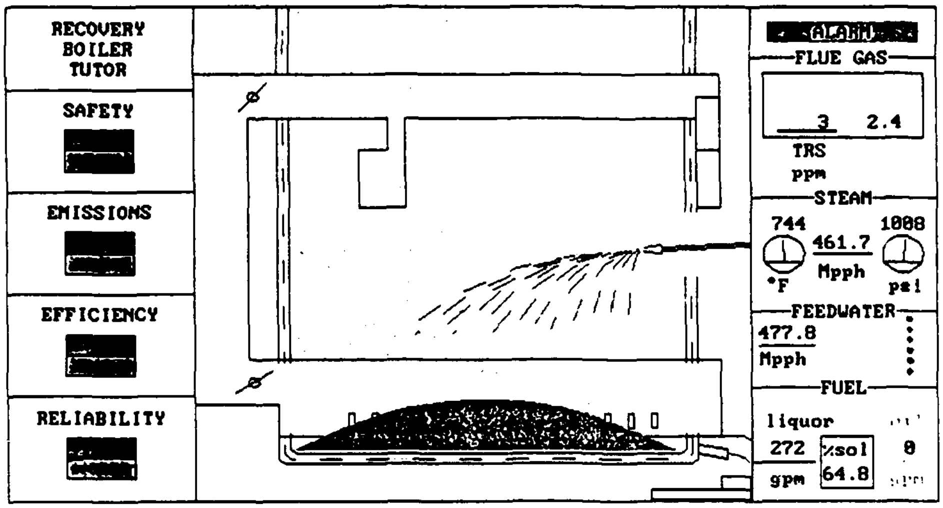


Figure 3. A focused view of the fire bed.

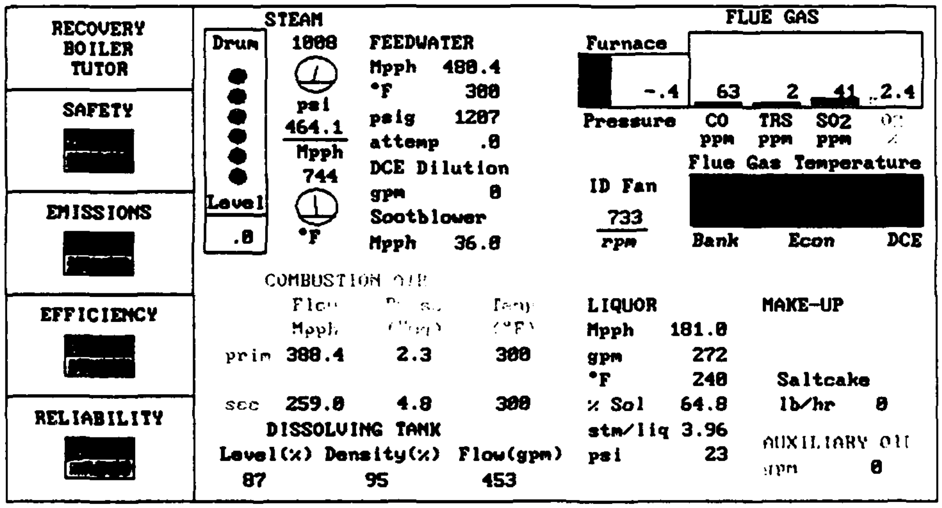


Figure 4. The complete control panel.

accumulated value could be used to activate more aggressive responses from the tutor.

RBT has been well received and is presently used in actual training in the control rooms of pulp and paper mills throughout the US. Formal evaluation has begun. However, informal evaluation suggests that operators enjoy the simulation and handle it with extreme care. They behave as they might in actual control of the pulp mill panel—slowly changing parameters, adjusting meters through small intervals, checking each action, and examining several meter readings before moving on to the next action. Experienced and novice operators alike engage in lively use of the system after about a half-hour introduction. When several operators interact with the tutor, they sometimes trade "war stories," advising each other about rarely seen situations. In this way, experienced operators frequently become partners with novice operators as they work together to simulate and solve unusual problems.

RBT was developed on an IBM PC AT with 512K-byte RAM, enhanced graphics, and a 20M-byte hard disk. It uses a math coprocessor, two display screens (one color), and a two-key mouse. The simulation was implemented in Fortran and took 321K bytes; the tutor was implemented in C and took 100K bytes. Although we tried to implement the tutor in Lisp, we found extensive interfacing and memory problems, including segment size restrictions (64k), incompatibility with the existing Fortran simulator, and addressable RAM restrictions (640K).

To circumvent these problems, the tutor was devel-

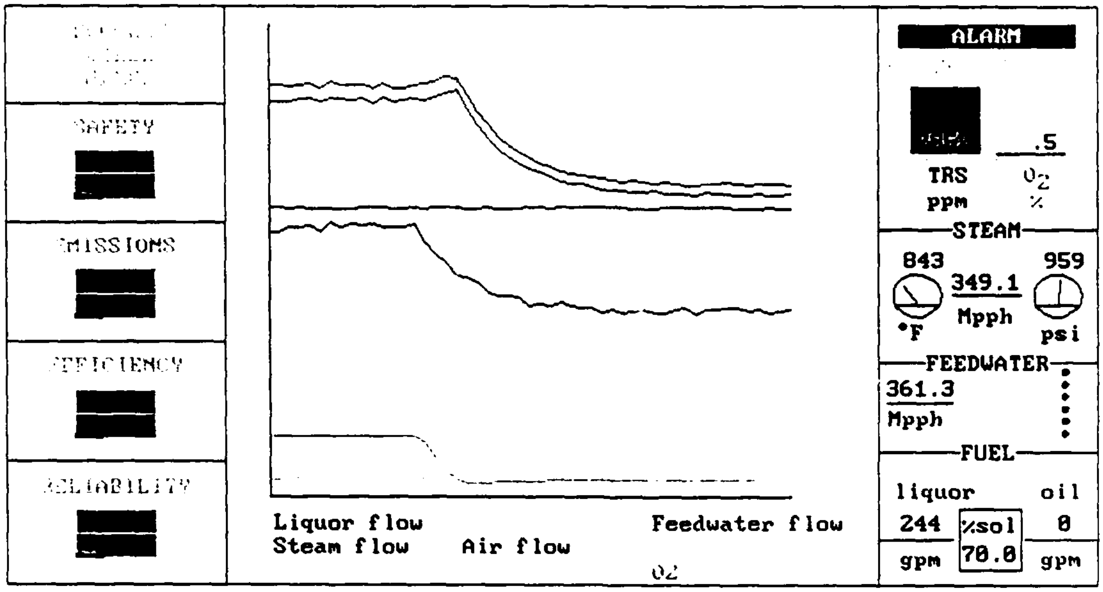


Figure 5. Trends selected by the operator.

oped in C, with many Lisp features implemented in C such as functional calls within the parameters of C functions. Meter readings and student interactions in the simulation were transferred between Fortran and C through vectors passed between the two programs.

Caleb for teaching a second language. Our second intelligent tutor teaches languages based on a powerful pedagogy called the "Silent Way"—a method developed by Caleb Gattegno that uses nonverbal communication within a controlled and artificial environment to teach second languages.[8] Learning a "second" language differs from acquiring a "first" language (as a child does). Children learning a first language must acquire linguistic competence such as the ability to associate meaning with sound, to make transformations, and to derive rules from observations and experimentation. Persons who have learned a first language have this linguistic competence and the Silent Way capitalizes on their ability by providing a directed environment engaging them in new, but analogous, linguistic experiences. The directed discovery environment engages students in an active learning process.

In Figure 6a, the tutor presents a new piece—a rod located in the center box. The student responds by typing the word for the new piece at the cursor. In Figure 6b, the student invents a new phrase by combining old pieces with the new one. In Figure 6c, the tutor corrects a student who places the adjective before (rather than after) the noun.

The tutor teaches Spanish by using graphical representations of Cuisenaire rods (originally developed by Gattegno for teaching arithmetic) to generate linguistic situations in which new words such as nouns, verbs, and adjectives are associated with meaning. On the screen, a rod is shown playing various roles; for example, it is used as an object to be given or taken by a student, or it is used to brush teeth. As a new rod is presented, students theorize about what situation is being represented, offer their conjectures, and revise their hypotheses as the situation demands. Students respond by typing a phrase to describe the situation in the text window. If the picture of a rod appears while the words *una regleta* are displayed, students type *una regleta* (as seen in Figure 6a). If students theorize that a new word such as *blanca* describes the size of the rod, they can later change that definition if (in fact) the new word defines the color of the rod (see Figure 6b). Meanwhile, students will have learned to write the word, spell it, and place it correctly in a sentence (see Figure 6c). They will have classified the word as a descriptor and will have invented phrases using it. Making and correcting hypotheses is central to language learning. Learner refinement of word meaning (that is, by a closer and closer approximation of expert ability) is one way to achieve linguistic mastery.

The tutor does not display words except to mention once, and only once, each new word in the target language. The tutor provides minimal *pieces* of the new language where *piece* is defined to be a phoneme, syllable, word, or phrase (see Figure 7). Pieces are aspects of the language that students can't invent, such as vocabulary and pronunciation. The tutor communicates silently, using icons, edit signals, and the rods—

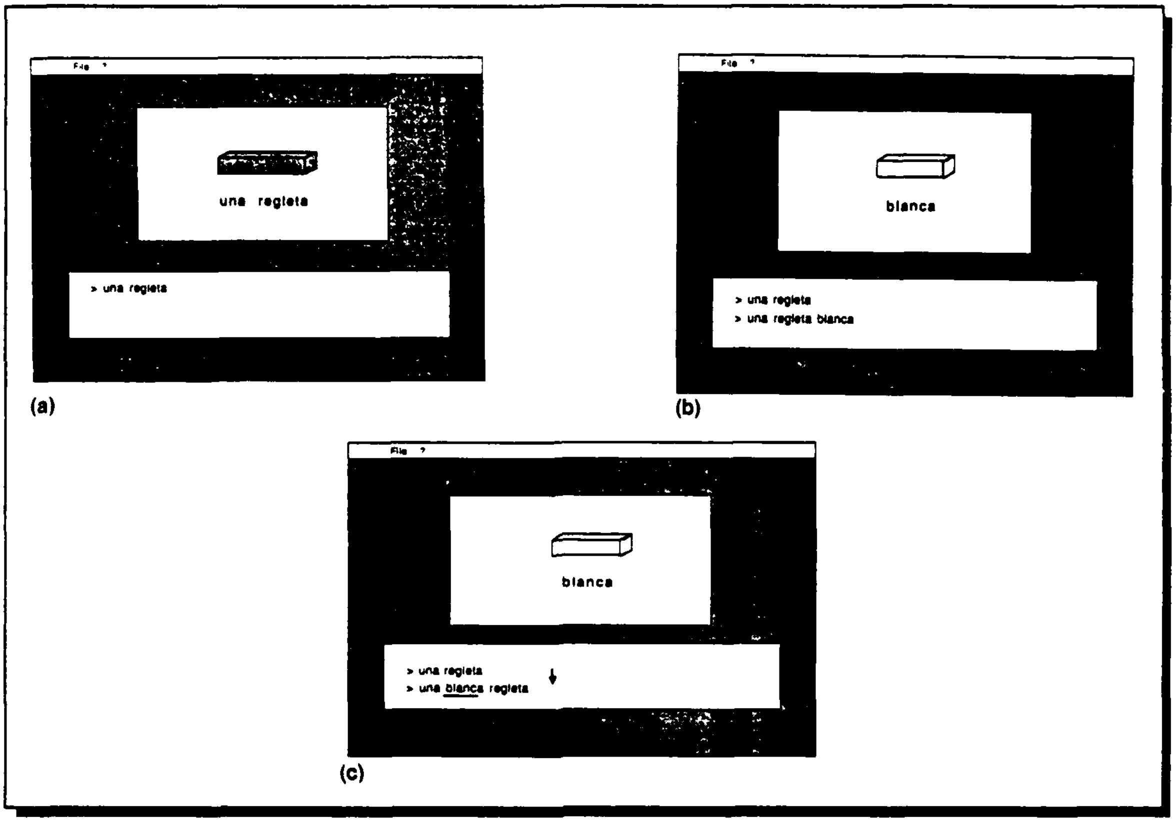


Figure 6. A system for teaching second languages.

Theme	Pieces	Potential misconceptions
1. Noun	a rod, una regleta	word order/agreement
***example response:** una regleta*		
2. Adj	white, blanca grey, gris striped, listada dotted, punteada	word order/agreement
***example response:** una regleta blanca*		
3. Conj	and, y	word order/use in series
***example response:** una regleta gris y una regleta blanca — una regleta gris, una regleta blanca, y una regleta negra*		
4. Numbers	two, dos -s three, tres -s one, una	word order/agreement
***example response:** dos regletas blancas y tres regletas negras*		
5. Noun deletion	one, blank ones	use in series/agreement
***example response:** una regleta roja y una blanca · dos regletas blancas y tres negras*		
6. Verb + direct object	take, toma	word order/agreement
***example response:** toma una regleta gris · toma una regleta gris y tres blancas*		
7. Verb + indirect object	give me, dame	word order/pronoun/agreement/case
***example response:** dame una regleta blanca · dame tres regletas negras y dos blancas*		
8. Pronoun	it/them, la/las	word order/pronoun/agreement/case
***example response:** toma una regleta blanca y damela · toma tres regletas negras y damelas*		

Figure 7. Themes and pieces in the Spanish curriculum.

Label	Idea to get across	Icon
go	it is your turn to do something	blinking cursor
wait	tutor busy, don't worry about nothing happening	Mac watch
attention	tutor about to do something new	sound
signal OK	let student know his response is OK	happy face
signal error	let student know he has error	sad face/Mr. Yuk
puzzled/repeat	say/do again (unintelligible response)	puzzled face/again sign
more	say/do more (incomplete response)	hand pull
help	help is available	?
dictionary	list of words already covered	book shape
throw out	extra stuff, do not need	Mac trash can
stop	save and quit = good bye	hand waving bye/stop sign
pause	take a break/stop timer	coffee cup
take	icon for mouse action	grasping hand
give	icon for mouse action	open hand
pacing regulator	slow down or speed up	speedometer
frustration gauge	student emotional state	thermometer
error correcting	word processing techniques	highlight/blinking cursor/placement arrows/fade in line

Figure 8. Communication icons in Caleb.

only "speaking" to provide words that students have not yet heard. Figure 8 lists icons used to represent these gestures, edit signals, and pantomime. These icons are used by the tutor, who plays the role of an orchestrator and monitor rather than that of an information giver.

With the introduction of verbs, action becomes possible. For example, the tutor prompts for a command by indicating a hand taking two white rods in the graphics window. The student types the command *Toma dos regletas blancas* ("Take two white rods") and the tutor performs the action.

So that students both produce and understand language, the tutor triggers two kinds of response: word-oriented responses typed at the keyboard, and action-oriented responses performed with the mouse and pictured objects. An example of the latter is the tutor giving the command *Toma dos regletas blancas* ("Take two white rods"). Students respond by using the mouse to take two pictured white rods with a grasping-hand-shaped cursor.

The non-verbal communication (that is, pantomime such as gestures, nods, and hand signals) of a Silent Way tutor are used to teach students to recognize when it is their turn to produce a sentence, when the tutor is about to say something, when the tutor expects more than students have produced, when an error needs correcting, or how to get help when they are stuck.

Errors are produced as students test and revise their theories about the language. When an error does occur, students are not deluged with entire sentences, nor are they provided a correct model to imitate. Instead, the precise location of the error is pointed out, as in Figure 6c, so that students may correct it themselves. The goal is to to let students develop their own sense of correctness or *inner criteria* for the new language.

The tutor monitors student input for correctness. A fault-tolerant parser filters "noisy" input so that some errors are ignored and some are treated, depending upon the situation. When the tutor treats an error, only the piece, noun, verb, or adjective that needs correcting is pointed out and students edit their input.

Regarding the presentation order of new material, the tutor makes decisions based on whether it's teaching the current piece, old piece, or old theme (see Figure 7). It uses five contexts to determine the number of times students should practice the piece. In the *intro context*, for example, the tutor simply presents the first example on the list of examples associated with each piece. When the tutor is in the *practice context*, the example source remains the same and the tutor marches down the list in a fixed order. In the *more practice context*, examples are chosen randomly from the piece example list. In the *review context*, examples are taken from an example pool of either the current theme or old themes. In the *error context*, examples come from the list associated with the error itself.

This system is implemented on a Macintosh and a Sun.

ESE for teaching thermodynamics. A third tutor built by Exploring Systems Earth—three universities working together to develop intelligent tutors—ESE is now in the early stages of implementation. This third tutor is one of a set of tutors that use interactive and monitored simulations for teaching elementary physics at high school and college levels. The goal is to put students in direct contact with physics elements such as mass, acceleration, and force. Students pursue various activities, such as changing the position or velocity of bodies in a celestial mechanics simulation, while viewing dependent changes in the size, speed, and position of orbit to improve their intuition about physical concepts. Each tutor monitors and advises students while providing examples, analogies, or explanations based on student actions, questions, or responses.

The tutor we are working on addresses the second law of thermodynamics—the law stating that heat cannot be absorbed from a reservoir and completely converted into mechanical work. This law is taught at the atomic level using a rich environment through which the principles of equilibrium, entropy, and thermal diffusion can be observed and tested.[9] Students are shown (and are able to construct) collections of atoms that transfer heat to one other through random collision (see Figure 9).

Students can create areas of high-energy atoms, indicated by dark squares, along with variously shaped regions within which the system can be analyzed and monitored. Several systems can be constructed, each with specific areas of high energy and associated observational regions. Concepts such as temperature, energy density, and thermal equilibrium for each observational region can be plotted against each other and against time. Students can then observe thermodynamic principles, such as heat transference through random collision and entropy as a function of initial system organization.

Students can modify system temperature, the number of collisions per unit time, and the shape of the initial "hot" region at any time. Changes in these parameters will cause dependent changes in the system. The tutor uses all student activities—including questions, responses, and requests—to formulate its next teaching goal and activity. It uses student actions to determine whether to show an extreme or near-miss example, whether to give an analogy or to ask a question. It bases its lessons on an ordered sequence of topics. To refine the tutor's response, we are now studying student misconceptions and common errors in learning thermodynamics and statistics.

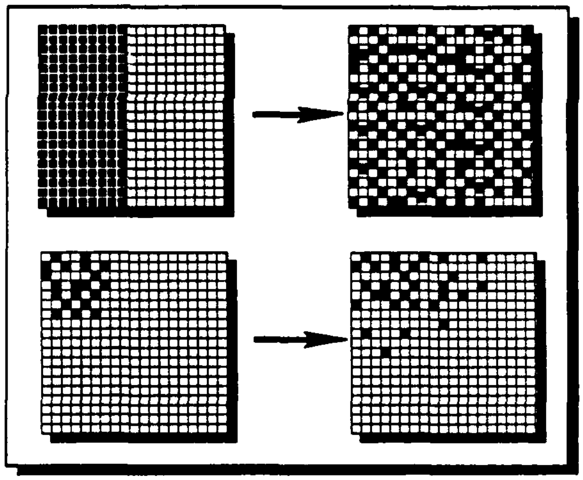


Figure 9. Systems moving towards equilibrium.

Klaus Schultz is ESE's domain expert, and Tom Murray the domain and teaching expert. Craig Lant, Paul Duquette, and Miguel de Campos are the environmental experts. The three universities comprising Exploring Systems Earth are the University of Massachusetts, San Francisco State University, and the University of Hawaii.

The need for multiple experts

Observations about the participation of experts in building these and other intelligent tutoring systems suggest that multiple experts must work on system design and implementation.[1,10] Much knowledge is distributed, subjective, unorganized, and misunderstood. No one human can supply the knowledge required—knowledge including environmental, teaching, cognitive, and domain expertise.

Given the complex and heterogeneous nature of knowledge required, we need methodologies and tools to transfer teaching and learning knowledge from each expert to the system under construction. Currently, few such tools exist.

Expert system shells contain a framework for building knowledge bases about concepts and rules, and for making inferences about them. Shells make building a tutor simpler to write.[11,12] However, we need more specific tools for designing and storing tutoring knowledge, and shells are limited in this respect. They are

frequently based on production rules and are limited in representing history and dependency of the tutoring interaction. Also, they inadequately represent tutoring and misconception knowledge—such as how to reason about teaching strategies, how to update and assess student models, how to select a path through domain concepts, and how to remediate for misconceptions.

The environmental expert. The first expert needed to build an intelligent tutor is the environmental expert. This person often uses a majority of system memory[1] to provide an envelope within which students and system interact. The environment provides specific tools and operators for solving domain problems or for performing domain activities.

Environmental, teaching, cognitive, and domain expert contributions interact strongly with each other—especially that from the environmental expert. For example, a system asking students to record entrance and exit angles for light rays in an optics experiment implies that the environment supplies such measuring devices.

The following criteria for developing tutoring environments have begun to emerge:

(1) Environments should be intuitive, obvious, and fun. Student energy should be spent learning the material, not learning how to use the environment.[13] For example, to indicate errors, express feelings, or convey meaning, the second language tutor's visual activities (see Figure 6) mimic the human Silent Way teacher's gestures, facial expressions, and rods. Each icon is designed to be clear, unambiguous, and to make use of student intelligence, experience, and resourcefulness.

(2) Environments should record not only what students do but what they intended to do, might have forgotten to do, or were unable to do.[14] Environments should provide a "wide bandwidth" within which multiple student activities can be entered and analyzed. For example, the Pascal tutor developed by Johnson and Soloway processed and analyzed an entire student program before offering advice.[15]

(3) Environments should be motivated by teaching and cognitive knowledge about how experts perform tasks and the nature of those tasks. For example, Anderson performed extensive research with geometry students before developing his geometry tutor interface,[16] and Woolf et al. incorporated knowledge from experts with more than 30 years experience working with boiler operations before building the RBT interface.[7]

(4) Environments should isolate key "tools" for attaining expertise in the domain. The economics tutor provided and monitored student use of key tools for performing economic experiments.[17] The RBT tutor provided process parameter graphs tracing trends over time, and used abstract meters to help operators reason about complex processes—thereby enabling them to make inferences about the effect of their actions (see Figures 2 through 5).

(5) Environments must maintain physical fidelity: Fidelity measures how closely simulated environments match the real world.[18] High fidelity identifies a system as almost indistinguishable from the real world. The RBT tutor presents a mathematically exact duplicate of the industrial process. It models and updates over 100 parameters every two seconds. Visual components of the industrial process such as alarm boards, control panels, dials, and reports are duplicated from the actual control room.

(6) Environments should be responsive, permissive, and consistent.[19] They should target applications based on skills people already have (such as moving icons) rather than forcing people to learn new skills. By responsive, we mean that student actions should have direct results—that students need not perform rigid sets of actions in rigid and unspecified order to achieve goals. By permissive, we mean that students may do anything reasonable and that multiple ways should exist for taking action. By consistent, we mean that moving from one application to another (for example, from editing text to developing graphics) should not require learning new interfaces. All tools should be based on similar interface devices, such as pull-down menus or single and double mouse clicks.

No environment is appropriate for every domain: We must study each domain to determine how experts function in that domain, how novices might behave differently,[20,21] and how novices can be helped to attain expertise.

The teaching expert. Acquiring sufficient and correct teaching expertise is a long-term problem for builders of tutoring systems—in part because sophisticated knowledge about learning, teaching, and domain knowledge remain active areas of research in most domains. For the machine tutor, designing decision logic and rules to guide tutorial interaction is a process of successive iteration—a process that can be improved continuously.

Tools to facilitate inclusion of long-term teaching knowledge are just beginning to emerge. For example,

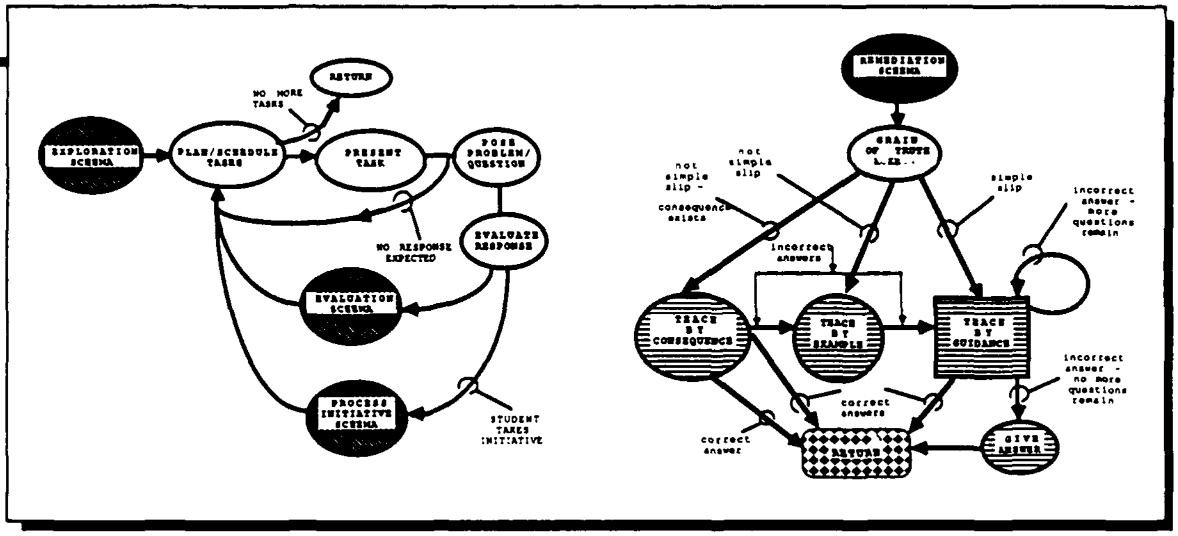


Figure 10. A framework for managing tutoring discourse.

we have developed a framework for managing discourse in an intelligent tutor that reasons dynamically about discourse, student response, and tutor moves (see Figure 10).[22] This is a flexible and domain-independent framework that is portable to various machine tutors. Also, it can be rebuilt—decision points and machine actions are modifiable for fine-tuning system response.

The framework reasons about which pedagogical responses to produce and which alternative discourse moves to make. It custom-tailors feedback to students in the form of examples, analogies, and simulations. Discourse *schemas* have been defined as collections of discourse activities and response profiles. Discourse is planned by passage through the schemas. The number and type of schemas selected depend on context.

We used empirical criteria to define these schemas: Tutoring responses were analyzed from empirical studies of teaching and learning,[23] other responses appeared as effective teaching strategies in research on misconceptions in physics,[24,25] others obeyed felicity laws, [26] and still others obeyed general rules of discourse structure.[27,28] General rules and fundamental remediation strategies have been incorporated into our discourse schemas and included in our teaching framework.

By continuously adjusting to real-time changes in either the knowledge base or the user, the discourse framework allows the tutor to remain flexible while cooperatively engaged in conversation. The framework views discourse as navigation through a set of possible discourse situations (see Figure 10). We are now using this framework to improve the physics tutor's response to idiosyncratic student behavior. The structure is preliminary in the sense that it's designed to be rebuilt; response decisions and machine actions, explicitly represented in the system, can be modified through a graphics editor. Appropriate machine response can be assessed and continuously improved through the editor. In the long term, we intend to make this reasoning process available to human teachers, who can then modify the tutor for use in a classroom.

No single teaching strategy is appropriate for every domain: Each domain and each student must be assessed to determine an appropriate teaching strategy. For example, Anderson et al. built geometry and Lisp tutors that respond immediately to incorrect student answers.[11] These authors argued that they needed immediate computer feedback to avoid fruitless student effort. They suggested that erroneous solution paths in geometry and Lisp might be so ambiguous and delayed that errors would not be recognized by students if tutor therapy were delayed.

This pedagogy is opposite to that used by Cunningham et al.[8] and Woolf et al.[7] These tutors were passive (not intrusive) advisors. Their strategy was to subordinate teaching to learning, allowing students to experiment while developing hypotheses about the domain. They guided students toward developing student intuitions, but did not correct students so long as student performance appeared to be attaining a precise goal.

In industrial settings, particularly, trainees must learn to generate multiple hypotheses and to evaluate their performance based on how their actions affect the industrial process. No human tutor is available during normal boiler operation. Thus, the machine's teaching strategy encouraged students to trust their own observations about the industrial process—

helping them to learn through animated simulations, trend analyses, and real-time, dynamically updated meters. In addition, textual dialogue (1) assured that operators would extract as much information as possible from data, and (2) established a mechanism to redirect them if they did not.

The cognitive expert. At present, the role of the cognitive scientist is incompletely understood; in part, this researcher seeks to discover how people learn and teach in a given domain. For example, cognitive science research in thermodynamics will enable systems to recognize common errors, tease apart probable misconceptions, and provide effective remediation. Cognitive science research provides the tutor with a basis for selecting instructional strategies. The importance of addressing common errors and misconceptions in physics is well documented, and the tutor's intelligence hinges on making that knowledge explicit.[24,25]

We want a tutoring system to help students (1) to generate those hypotheses that are necessary precursors to expanding their intuition, and (2) to develop their own models of the physical world, while discovering and "listening to" their own scientific intuitions. To do this, we rely on work done by cognitive scientists, who study how students reason about qualitative processes, how teachers impart propaedeutic principles (or the knowledge needed for learning some art or science),[29] and what tools are being used by experts working in the field.

For example, the cognitive science experiments that must be performed to build our thermodynamics tutor include (1) investigation of real-world tools currently used by physicists, (2) examination of studies that focus on cognitive processes used by novices and experts, and (3) comparison of novice with expert understanding of physics problems. Cognitive science can define such knowledge—knowledge that elucidates actions taken by experts to make measurements or perform transformations in a domain. We call this "heuristic knowledge" and define it as *knowledge about how to solve domain problems.* Heuristic knowledge differs from procedural knowledge in that it adds neither content nor concepts to the domain, but describes actions taken by experts using their conceptual and procedural knowledge. Such knowledge, rarely included in tutoring systems, must be included if tutors are to monitor student problem-solving activities and experiential knowledge about how to work in the field.

RBT articulates this knowledge by explicitly recording student attempts to solve emergencies. It shows students their false paths and gives reasons behind particular rule-of-thumb knowledge used to solve problems. RBT also provides students with various examples from which they can explore problem-solving activities—showing students their own paths, preferred paths, and (perhaps, in time) their own underlying cognitive processes. Simply elucidating these operational problem-solving components in a domain, and the rules applying to their use, is not sufficient to understand how one learns in a new domain. By using such knowledge, however, a tutor can help students *learn how to learn.* We are compiling such data and expect that it (along with cognitive studies) will elucidate some processes behind problem-solving behavior.

The domain expert. An in-house domain expert is a critical requirement for building intelligent tutoring systems. By "in-house," we mean that the domain expert must join the project team for anywhere from six months to several years while domain knowledge is being acquired. Less commitment than that—that is, any role less than that of full-fledged team member—suggests a less-than-adequate transfer of domain knowledge.

In the tutors described above, domain experts were (and are) integral to the programming effort. The programmer, project manager, and director of RBT were themselves chemical engineers. More than 30 years of theoretical and practical knowledge about boiler design and teaching strategies were incorporated into the system. Development time for this project would have been much longer than 18 months if these experts had not previously identified the boiler's chemical, physical, and thermodynamic characteristics and collected examples of successful teaching activities.

An instructor holding an ESL graduate degree (English as a second language) developed the second-language tutor. This instructor (the second author of this article) has more than seven years teaching experience. She has used the Silent Way to teach intensive English courses to foreigners living in America, and to train Nepali language instructors who in turn taught Nepali to future American Peace Corps volunteers.

The physics tutor is being built after more than 18 months of work with member physicists and astronomers. In addition, potential users of the system (high school and college physics teachers) are contributing environmental and teaching knowledge. Their overall effort produced more than 100 pages of rules, processes, screen designs (including help activi-

ties about physics), and cognitive studies (identifying educational goals, potential errors and misconceptions) before any code was built.

For domain knowledge, expert shells can be used effectively. Anderson[11] used Grapes[30] to represent the rules programmers use for solving problems, to describe Lisp functions, and to represent higher level programming goals. He used *buggy* rules to represent misconceptions that novice programmers often develop during learning. Streibel et al.[12] used OPS5 to write rules for genetic problem solving and to encode teaching strategies.

Based on the various expert systems that have been built, the following criteria for acquiring domain knowledge are well understood:

(1) Domain experts should be true experts—if possible, the best in the field.[1] Dendral, for example, an expert system for generating and testing hypotheses about chemical structures and spectroscopic data, was built by a team including Joshua Lederberg (a Nobel-prize-winning geneticist), Carl Djerassi (a world-class expert on mass spectral analysis), and other professional chemists and computer scientists.[31]

(2) Domain experts are expensive. Gaining the attention of knowledgeable people in any domain is expensive and time consuming. However, the willingness and availability of such experts to participate is critical to the knowledge-engineering process. Assigning the expert role to someone of lesser ability (or worse, to persons with "time on their hands") might doom a project to failure. On the other hand, enthusiastic support from funders and supervisors—including sufficient allocation of resources, human and otherwise—are prerequisite to success.

(3) Certainly, individual domain experts can have incomplete knowledge or conceptual vacuums. Multiple experts are needed for testing and modifying domain knowledge throughout the tutor's life.

(4) Similarly, domain knowledge can be overly distributed.[1,10] That is, knowledge can be spread so diffusely among different research projects and experts as to leave any system unfinishable that uses only a single expert (or even several experts). Thus domain knowledge must be acquired incrementally and must be prototyped, refined, augmented, and reimplemented. The time needed to build a tutoring system "should be measured in years, not months, and in tens of worker years, not worker months."[1]

(5) Domain knowledge found in textbooks is incomplete and idealized.[1] In an intelligent tutoring system, textbooks are inappropriate as a primary source for either domain or teaching knowledge. Textbooks rarely contain the commonsense knowledge—the know-how used by expert tutors or professionals in the field—to help choose a next-teaching strategy or solve difficult problems. Books tend to present clean, uncomplicated concepts and results. To teach or solve real-world problems, tutors must know messy but necessary details of real and perceived links between concepts and unpublished rules of teaching and learning.

Intelligent tutors can neither cure all educational problems nor totally answer the dilemma facing education. They seem incapable of achieving the very difficult, let alone the impossible. However, they do provide exciting possibilities—and of these, one of the most exciting is that of providing a community memory for teaching and learning research. This community memory would provide a focus for articulating distributed knowledge in an intelligent tutor. It would include recent as well as historical research about thinking, teaching, and learning. Evaluating such an articulation would, in itself, contribute to education—and ultimately, to communication between experts.

Compiling diverse research results from environmental, teaching, cognitive, and domain experts is currently hampered by the lack of explicit methodologies and technologies to help authors transfer their knowledge to a system. This article has specifically addressed the issues of what further knowledge we need and where we should apply human and financial resources to build future intelligent tutoring systems. ■

Acknowledgments

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York—and by the Air Force Office of Scientific Research, Bolling AFB, the District of Columbia—under contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium.

References

1. D. Bobrow, S. Mittal, and M. Stefik, "Expert Systems: Perils and Promise," *Comm. ACM*, Sept. 1986.
2. J. McDermott, "A Rule-Based Configurer of Computer Systems," *Artificial Intelligence*, Jan. 1982.
3. R. Smith, "On the Development of Commercial Expert Systems," *AI Magazine*, Fall 1984.
4. D. Waterman, *Innovation: The Attacker's Advantage*, Simon & Schuster, New York, N.Y., 1986.
5. D. Bitzer, D. Braunfeld, and W. Lichtenberger, "PLATO: An Automatic Teaching Device," *IRE Trans. Education*, Dec. 1961, pp. 157-161.
6. A. Bork, "Physics in the Irvine Educational Technology Center," *Computers in Education*, No. 4, 1980, pp. 37-57.
7. B. Woolf et al., "Teaching a Complex Industrial Process," *Proc. Am. Assoc. Artificial Intelligence*, Philadelphia, Pa., Aug. 1986.
8. P. Cunningham, T. Iberall, and B. Woolf, "Caleb: An Intelligent Second Language Tutor," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics*, Computer Soc. IEEE, Los Alamitos, Calif., 1986, pp. 1210-1215.
9. T. Atkins, *The Second Law*, Freedman, San Francisco, Calif., 1982.
10. S. Mittal and C. Dym, "Knowledge Acquisition from Multiple Experts," *AI Magazine*, Summer 1985.
11. J. Anderson, C. Boyle, and G. Yost, "The Geometry Tutor," *Proc. Ninth Int'l Joint Conf. Artificial Intelligence*, Los Angeles, Calif., Aug. 1985.
12. M. Streibel et al., "An Intelligent Computer Tutoring System for Solving Genetics Problem Solving, Conjecturing, and Understanding," *Machine-Mediated Learning*, Vol. 2, Nos. 1 and 2, 1987, pp 129-159.
13. P. Cunningham, *Caleb: A Silent Second-Language Tutor: The Knowledge Acquisition Phase*, Tech. Report #87-6, Rensselaer Polytechnic Institute, Troy, N.Y., 1986.
14. R. Burton, "Instructional Environments," in *Foundations of Instructional Tutoring Systems*, M. Polson and J. Richardson, eds., Lawrence Earlbaum Associates, Hillsdale, N.J., in press.
15. L. Johnson and E. Soloway, "Intention-Based Diagnosis of Programming Errors," *Proc. Am. Assoc. Artificial Intelligence*, Austin, Tex., Aug. 1984.
16. J. Anderson, "Tuning of Search of the Problem Space for Geometry Proofs," *Int'l Joint Conf. Artificial Intelligence*, British Columbia, Canada, Aug. 1981.
17. V. Shute and J. Bonar, "Intelligent Tutoring Systems for Scientific Inquiry," *Proc. Cognitive Science Soc.*, Lawrence Earlbaum Associates, Hillsdale, N.J., 1986.
18. J. Hollan, E. Hutchins, and L. Weitzman, "Steamer: An Interactive Inspectable Simulation-based Training System," *AI Magazine*, Summer 1984.
19. Apple Corp., *Inside Macintosh*, Vol. 1, Addison-Wesley, Reading, Mass., 1985.
20. W. Chase and H. Simon, "The Mind's Eye in Chess," in *Visual Information Processing*, W.G. Chase, ed., Academic Press, New York, N.Y., 1973.
21. J. Larkin et al., "Expert and Novice Performance in Solving Physics Problems, *Science*, June 1980.
22. B. Woolf and T. Murray, "A Framework for Representing Tutorial Discourse," to appear in *Proc. Int'l Joint Conf. Artificial Intelligence*, Milan, Italy, Aug. 1987.
23. J. Clement and D. Brown, "Using Analogical Reasoning to Deal with Deep Misconceptions in Physics," tech. report, Cognitive Research Group, Physics Department, University of Mass., May 1984.
24. J. Clement, "Students' Preconceptions in Introductory Mechanics," *Am. J. Physics*, Jan. 1982.
25. L. McDermott, "Misconceptions in Classical Mechanics," *Physics Today*, July 1984.
26. K. van Lehn, "Felicity Conditions for Human Skill Acquisition: Validating an AI Theory," research report, Xerox Palo Alto Research Center, Palo Alto, Calif., 1983.
27. B. Grosz and C. Sidner, "The Structure of Discourse Structure," *Proc. Am. Assoc. Artificial Intelligence*, Los Angeles, Calif., Aug. 1985.
28. R. Reichman-Adar, *Plain Speaking: A Theory and Grammar of Spontaneous Discourse*, PhD dissertation, Harvard University, Department of Mathematics, Cambridge, Mass., 1981.
29. H. Halff, "Curriculum and Instruction in Automated Tutors," in *Foundations of Instructional Tutoring Systems*, M. Polson and J. Richardson, eds., Lawrence Earlbaum Associates, Hillsdale, N.J., in press.
30. R. Sauers and R. Farrell, "Grapes User's Manual," tech. report, Office of Naval Research 82-3, Carnegie Mellon University, Pittsburgh, Pa., 1982.
31. R.K. Lindsay et al., *Applications of Artificial Intelligence for Organic Chemistry*, William Kaufmann, Los Altos, Calif., 1985.

Beverly Woolf is an assistant professor of computer and information sciences at the University of Massachusetts. Her research interests include AI, cognitive science, knowledge representation, and human-computer interfaces for explanation and teaching. She and her research group have active projects in machine tutoring, student modeling, discourse management, and intelligent interfaces. She received her BA in physics from Smith College, and her MS and PhD in computer science from the University of Massachusetts.

Pat Cunningham is a systems programmer in the electrical engineering department at Yale University. She has taught mathematics, science, and English as a second language in Nepal and in the US. Her current research focuses on using computers in educational settings, especially intelligent tutoring systems. She received her BA in Spanish from Thiel College, her MA in ESL from the University of Hawaii, and her MS in computer science from Rensselaer's Hartford Graduate Center.

The authors can be reached in care of Beverly Woolf, at the Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

Representing complex knowledge in an intelligent machine tutor

BEVERLY P. WOOLF
Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003, U.S.A.

Received August 18, 1986
Revision accepted January 28, 1987

Knowledge representation remains a serious issue for researchers of intelligent tutoring systems. Two areas of knowledge representation that are particularly difficult are domain and teaching knowledge. This article discusses and gives example solutions to these knowledge engineering issues and also addresses issues that relate to up-scaling existing intelligent tutoring technology to practical levels so that tutoring systems can be brought into the real world.

Key words: intelligent tutoring systems.

La représentation de connaissances pose toujours des problèmes sérieux aux chercheurs travaillant sur les systèmes tuteurs intelligents. Deux domaines de représentation de connaissances particulièrement difficiles sont ceux liés à la connaissance du domaine et à la connaissance pédagogique. Cet article discute ces problèmes de génie cognitif en apportant quelques exemples de solutions. Il s'intéresse enfin aux problèmes à résoudre pour amener la technologie des systèmes tuteurs intelligents à un niveau pratique suffisant pour qu'ils sortent du laboratoire.

Mots clés: systèmes tuteurs intelligents.

[Traduit par la revue]

Comput. Intell. 3, 45–55 (1987)

1. Introduction

Intelligent teaching systems are designed to represent both the concepts to be taught to the student and how a student might learn those concepts. Intelligent tutoring systems clearly differentiate the components of the teaching process; thus, knowledge of the student (Johnson and Soloway 1984; Miller 1982) is separated from knowledge of the domain (Stevens *et al.* 1982) and both of these types of knowledge are separated from strategies about how and what to teach (Clancey 1982; Woolf and McDonald 1984*a,b*). Given this wealth of knowledge, often encoded in the form of hundreds of "if–then" rules, such systems perform fine-grained reasoning about a student and his progress. Codification of this tacit knowledge and transmission of this knowledge to the computer is what enables the intelligent tutor to represent what the student knows and to provide guidance in a form such that the student remains in control of the interaction. A distinction is also often made between "strong" and "weak" teaching systems. A strong teaching system solves the same problem it presents to the student and a weak one does not. Thus, in addition to recognizing errors, recording missing answers, and correcting errors, which a "weak" system can do, the strong system additionally solves the problem in order to be able to assist the student when he gets "stuck" during a partial solution. To do this, a strong teaching system needs additional "expert" system knowledge in addition to the knowledge listed above.

This paper provides example solutions to two knowledge representation issues: domain and teaching knowledge. The paper demonstrates how complex knowledge and increased research into teaching and learning are necessary to make further progress in the development of intelligent teaching systems.

2. Knowledge of the domain

Historically, the first knowledge issue addressed by researchers of intelligent tutors was knowledge of the domain. As tutoring systems developed, the more powerful systems required more sophisticated knowledge of the domain. For example, lists of concepts and rules, such as those found in textbooks, were insufficient to enable a system to observe errors, assist in recognizing misconceptions, and provide custom-tailored remedial action. Careful representation of domain knowledge required at least an investigation into an expert's understanding of the laws of the domain, possibly divided into classifications such as concepts, procedural rules, and meta-rules by which the concepts were used, heuristics (or rules of thumb) to use the rules, and simulation rules to graphically implement the data. This breakdown is not the "best" nor the only way to parcel domain knowledge, but it serves as an indication of the complexity of knowledge required in building an intelligent tutoring system.

To illustrate the complexity of domain knowledge in a strong intelligent tutor, we describe the Recovery Boiler Tutor (RBT), a tutor built for a kraft recovery boiler, which is a type of boiler found in paper mills throughout the United States (Woolf *et al.* 1986).[1] RBT provides multiple explanations and tutoring facilities tempered to the individual user, a control room operator. The tutor is based on a mathematically accurate formulation of the boiler and provides an interactive simulation complete with help, hints, explanations, and tutoring (Fig. 1).[2]

---

[1]RBT was built by J.H. Jansen Co., Inc., Steam and Power Engineers, Woodinville (Seattle), Washington and sponsored by The American Paper Institute, a nonprofit trade institution for the pulp, paper, and paperboard industry in the United States, Energy Materials Department, 260 Madison Avenue, New York, NY, 10016.

[2]RBT was developed on an IBM PC AT (512KB RAM) with enhanced graphics and a 20MB hard disk. It uses a math coprocessor, two display screens (one color), and a two-key mouse. The simulation was implemented in Fortran and took 321KB; the tutor was implemented in C and took 100KB. Although we tried to implement the tutor in Lisp, we found extensive interfacing and memory problems, including segment size restrictions (64K), incompatibility with the existing Fortran simulator, and addressable RAM restrictions (640K). To circumvent these problems the tutor was developed in C with many Lisp features implemented in C, such as functional calls within the parameters of C functions. Meter readings and student interactions in the simulation were transferred between Fortran and C, through vectors passed between the two programs.

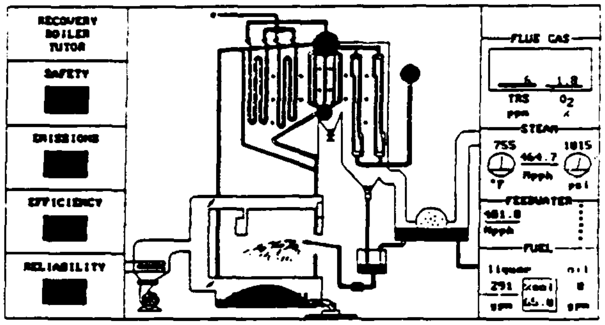


FIG. 1. *Selected view of the recovery boiler.*

A student can initiate any of 20 training situations, emergencies, or operating conditions, or he can ask that an emergency be chosen for him. He can also accidently trigger an emergency as a result of his actions on the boiler. Once an emergency has been initiated, the student is encouraged to adjust meters and perform actions on the simulated boiler to solve the emergency.

The goal in building the system was to challenge the student operator to solve new problems while monitoring and commenting upon his actions. The system can recognize optimal, less than optimal, and clearly irrelevant actions. The operator continues his freewheeling or purposeful problem-solving behavior while the tutor offers help, hints, explanations, and tutoring advice when needed or requested. The operator is expected to observe the impact of his actions on the simulated boiler and to react before the tutor advises him about potential problems.

An example interaction[3] between the student and the tutor is shown in Fig. 2. As the operator changes setpoint controllers and requests information about the boiler, the tutor selectively discusses the optimality of his actions (we show how below) and suggests how he might better focus his action or better utilize his data. An important feature to note about this dialogue is that at any point during the simulated emergency there are a large number of actions an operator might take and, as the situation worsens, an increasing number of actions that he *should* take to correct the operating condition. Thus, an immediate and correct response might require only one action, such as rodding the primary air ports, but a delayed response causes the situation to worsen and requires the addition of auxiliary fuel.

The operator's interactions with the tutor are made through a hierarchy of menus, two of which are shown in Fig. 3. Menu A allows an operator to select a physical activity to be performed on the boiler, such as checking for a tube leak or rodding the smelt spout. Menu B allows the operator to select a particular computer screen, such as the alarm board or control panel board.

While the simulation of the recovery boiler is running, the operator can view the boiler from many directions and can focus on several components, such as the fire bed illustrated in Fig. 4. The tutor provides assistance through visual clues (such as a darkened smelt bed), acoustic clues (ringing alarm buzzers), textual help, explanations, and dialogues (such as that illustrated in Fig. 2). The operator can request up to 30 process parameters on the complete panel board (Fig. 5), view an alarm board (not shown), change 20 setpoints, and ask menued questions such as "What is the problem?", "How do I get out of it?", "What caused it?", and "What can I do to prevent it?".[4] The operator can request meter readings, physical and chemical reports, and dynamic trends of variables. All variables are updated in real time (every 1 or 2 sec).

[Note: A partial blackout is in progress and it will lead to deterioration *of the combustion process if no action is taken.* Meters indicate that CO, SO_2, and O_2 readings are abnormally high.]

[An alarm buzzer sounds.]

Tutor: What action are you going to take?

Student: Check smelt spout cooling water. [Student's action does not *result in a change in the boiler.*]

Tutor: It looks as though the liquor isn't burning...

Student: Help. What is going on? [Student requests help.]

Tutor: A partial blackout is occurring and this causes TRS, CO, SO_2, and O_2 to increase. Steam flow and steam temperature are dropping. I think part of the bed is not burning

Student: Rod the primary air ports. [A small improvement in the boiler readings can be observed.]

Tutor: *Your action is correct; however, at this time rodding alone is* not effective...

Student: Raise the liquor temperature. [Student's action does not reverse the emergency and a new alarm buzzer sounds.]

Tutor: Rodding the air port apparently was not enough to undo the blackout.

Student: Increase the oil fuel supply. [Student's action reverses the emergency.]

Tutor: That solved the problem alright. Good thinking.

Analysis of the problem: You had a partial blackout caused by plugged primary air ports and a cold bed. Partial blackout can be effectively treated through a combination of rodding the primary air ports and adding more heat. The problem can be avoided by keeping the air ports clean.

FIG. 2. *Dialogue between tutor and operator.*

In addition to providing information about the explicit variables in the boiler, RBT provides reasoning tools designed to aid a student in reasoning about implicit processes in the boiler. One such tool is composite meters (left side of Fig. 1 and 4 through 6), which records the state of the boiler using synthetic measures for *safety, emissions, efficiency*, and *reliability* of the boiler. The meter readings are calculated from complex mathematical formulae that would rarely, if ever, be used by an operator himself to evaluate the boiler. For instance, the safety meter is a composition of seven independent parameters, including steam pressure, steam flow, steam temperature, feedwater flow, drum water level, firing liquor solids, and combustibles in the flue gas. Meter readings allow a student to make inferences about the effect of his actions on the boiler using characteristics of the running boiler.[5]

Other reasoning tools include trend analyses, see Fig. 6, and animated graphics, such as shown on boiler figures. Trend

[3]The dialogue of Fig. 2 was not actually produced in natural language: student input was handled through menus (Fig. 3) and tutor output produced by cutting text from emergency-specific text files loaded when the emergency was evoked.

[4]These four questions are answered by cutting text from a file which was loaded with the specific emergency. These questions do not provide the basis of the tutor's knowledge representation, which will be discussed below.

[5]These meters are not presently available on existing pulp and paper mill control panels; however, if they prove effective as training aids, they could be incorporated into actual control panels.

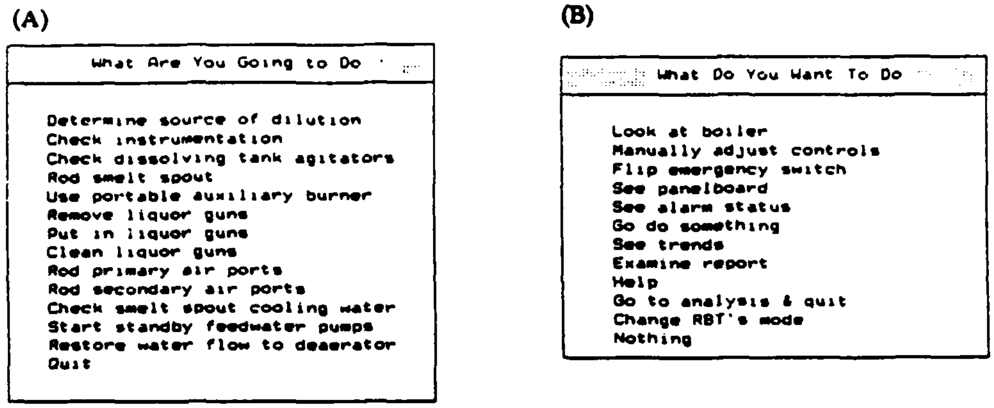


FIG. 3. Menus to select tasks to be performed on the boiler.

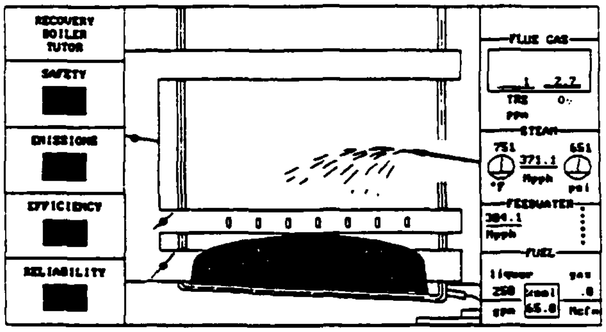


FIG. 4. Focused view of the fire bed.

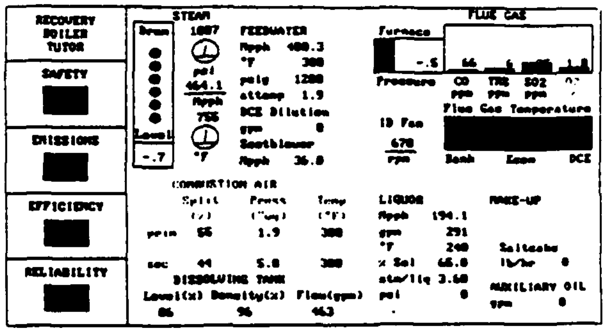


FIG. 5. The complete control panel.

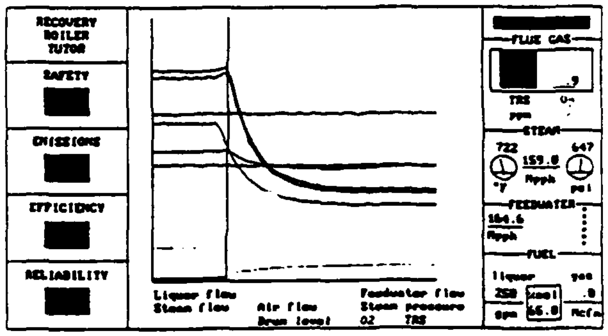


FIG. 6. Trends selected by the operator.

analyses show how essential process variables interact in real time by allowing an operator to select up to 10 variables, including liquor flow, oil flow, air flow, etc., and to plot each against the others and against time. Animated graphics provide realistic and dynamic drawings of the several components of the boiler, such as steam, fire, smoke, black liquor, and fuel.

Student actions are recorded in an accumulated response value, reflecting an operator's overall score and how successful, or unsuccessful, his actions have been and whether actions were performed in sequence with other relevant or irrelevant actions.[6]

2.1. *Multiple representations of domain knowledge*

In order to describe RBT, we represented several facets of domain knowledge, including concepts, rules, and learning strategies specific to the domain. Such knowledge was specified in excruciating detail before the tutor was built. This knowledge is divided into three categories: conceptual, procedural, and heuristic knowledge.

Conceptual knowledge includes data, concepts, and relation between concepts in the domain. This knowledge has traditionally been the primary domain knowledge represented in a tutoring system. In many systems, concepts are represented by a frame or other data structure that encodes default values within an explicit set of attributes for each concept. Such a data structure expresses information about both the attributes of a concept and the relationship between concepts.

Procedural knowledge includes the reasoning used to solve problems in the domain. This knowledge has traditionally been included only in teaching systems that reason about procedural tasks, such as solving arithmetic problems (Burton and Brown 1982) or simulating the operations of a steam engine (Forbus and Stevens 1981) and has typically been missing from systems based on simulations. In fact, as recognized by Hollan *et al.* (1984) and Forbus and Stevens (1981), simulations must also incorporate tutoring and must explain the qualitative processes behind the physical process in the simulation.

Heuristic knowledge includes actions taken by an expert to

[6]This accumulated value is not presently used by the tutor, but the notation might be used to sensitize the tutor's future responses to the student's record. For instance, if the operator has successfully solved a number of boiler emergencies, the accumulated value might be used to temper subsequent tutoring so that it is less intrusive. Similarly, if a student's past performance has been poor, the accumulated value cannot be used to activate more aggressive responses from the tutor.

make measurements or perform transformations in the domain. It defines those operations used to solve problems in the field and is part of an expert's experiential knowledge about the domain. Such knowledge differs from procedural knowledge in that it does not add content to the domain, nor does it actually access content in the domain; rather, it adds knowledge about *how to solve problems* in the domain and therefore describes actions taken by the expert while using conceptual and procedural knowledge. This knowledge has rarely been included in tutoring systems, but must begin to be included if tutors are to monitor their student's problem-solving activities.

2.2. Learning how to learn

Using heuristic knowledge can begin to help a student *learn how to learn*. A tutor with heuristic knowledge can show false paths taken by the student and can begin to give reasons behind the particulars of rule-of-thumb knowledge used to solve problems. For example, a tutor can provide with a variety of examples from which a student can explore a large space of problem-solving activities. The student's path through the activities and strategies can be traced; so the tutor can begin to define properties of the underlying cognitive process. We cannot derive deep properties of learning and problem solving simply by elucidating the steps taken to solve problems. Nevertheless, if we compile problem-solving data, along with cognitive studies, we can begin to elucidate some processes underlying a student's problem-solving behavior.

Philosophers of science believe that heuristic knowledge, unlike conceptual and procedural knowledge, is best acquired through experience and working examples illustrating aspects of the phenomenon (Kuhn 1970). Educators and cognitive scientists have observed that students benefit from numerous hours spent solving problems. Yet, there is nearly a total absence of information about which strategies and heuristics actually work, how a student learns from doing homework, or what precisely is learned from doing homework (see Larkin (1982) for some innovative studies along this line).

The computer helps explain the process of learning from examples by recording the student's behavior when confronted with examples of increasing complexity which are also linked by cognitive similarities. For example, RBT provides tools with which a student can build his own example emergencies. It simulates the newly made examples to "work" or "fail" according to the laws of the science domain. The student's ability to use rules of science in the simulation will in part predict his ability to work outside the simulation and within the complex industrial site.

Kuhn states

> A student cannot, it is said, solve problems at all unless he has first learned the theory and some rules for applying it. Scientific knowledge [it is said] is embedded in theory and rules; problems are supplied to gain facility in their application. I have tried to argue, however, that this localization of the cognitive content of science is wrong. After the student has done many problems, he may gain only added facility by solving more. But at the start and for some time after, doing problems is learning consequential things about nature. In the absence of such examples, the laws and theories he has previously learned would have little empirical content [Kuhn 1970; emphasis mine].

Kuhn (1970) points out that scientific breakthroughs often result from a scientist's need to solve unique problems for which no existent rule applies. Given anomolous data, a scientist will "retrofit" existent formulas, consider alternative solution paths, and, if possible, create new procedures to explain the phenomenon. Such techniques of modifying existent rules should be taught to students. Kuhn suggests that learning to recognize, apply, and reject operant (procedural) laws of nature is a prerequisite for "doing" science.

Because it is involved with strategies for solving a problem, heuristic knowledge must also point to the prerequisite knowledge that a student should know before he can solve the problem. Such "preknowledge," and the way the machine investigates it, is contained in the student model, which will be briefly discussed in the next section.

2.3. Domain knowledge in RBT

In RBT, concepts and processes were represented in multiple ways, some procedurally, some declaratively, and some in both ways. To a small degree heuristic knowledge was also represented. For example, emergencies in the steam boiler were first represented as a set of mathematical formulae so that process parameters and meter values could be produced accurately in the simulation. These same emergencies were then encoded within the tutor's knowledge base as a framelike data structure with slots for preconditions, optimal actions, and conditions for solution satisfaction so that the tutor could evaluate and comment upon the student's solution.

RBT can recognize and explain

- equipment and process flows,
- emergencies and operating problems as well as normal operating conditions,
- solutions to emergencies and operating problems,
- processes for implementing solutions, and
- tutoring strategies for assisting the student.

This knowledge was organized into four modules: *simulation, knowledge base, student model*, and *instructional strategies*.

The simulation uses a mathematical foundation to depict processes in a boiler through meter readings and four animated views of the boiler. It reacts to more than 100 process parameters and generates dynamically accurate reports of the thermal, chemical, and environmental performance of the boiler (not shown) upon request. An alarm board (not shown) represents 25 variables whose button will turn red and alarm sound when an abnormal condition exists for that parameter. The simulation is interactive and inspectable in that it displays a "real time" model of its process, yet allows the student to "stop" the process at any time to engage in activities needed to develop his mental models (Hollan *et al.* 1984). The operators who tested RBT mentioned that they liked being able to stop the process to ask questions or explore boiler characteristics.

If a student working on a problem inadvertently triggers a second problem, the least serious problem, as defined by the engineer, will be placed on a stack and held in abeyance while the student is coached to solve the more serious problem. After the more serious problem is solved, the student is coached to solve the remaining one. Thus, the simulation provides facilities for handling multiple instantiations of emergencies.

One advantage of a formal representation of the process is the availability of a "database" of possible worlds from which information based on typical or previous moves can be fed into the simulation at anytime (Brown *et al.* 1982) and a solution found. In this way, a student's hypothetical cases can be proposed, verified, and integrated into his mental model of the boiler.

The knowledge base contains procedural knowledge about

emergencies, called *scenarios*, and it includes preconditions, postconditions, and solutions for each. Scenarios are represented in framelike text files. For example, in Lisp notation, a true blackout would be described as

```
preconditions:
  (or (<= blackout_factor 1)
      (< heat_input 5000))
postconditions:
  (or (increasing O2)
      (decreasing steamflow)
      (increasing TRS)
      (increasing CO)
      (increasing SO2))
solution satisfaction:
  (and (= blackout_factor 1)
       (> heat_input 5200))
```

Scenarios in RBT represent successively more serious problems. For instance, a smelt spout pluggage is represented in terms of several scenarios depending on whether the solution requires rodding the spout, applying a portable auxiliary burner, removing the liquor, or a combination of all three. Again, formalized knowledge of the domain made it easy to represent and evaluate graduate scenarios, as well as multiple operator actions.

The efficiency of the student's action is evaluated both through the type of action performed, such as whether the student *increased* O_2 or *increased steamflow* for a true blackout, and the effect of that action on the boiler. Thus, if an inappropriate action nevertheless resulted in a safe boiler, the student would be told that his action worked, but that it was not optimal. For example, a partial furnace blackout requiring manual rodding of the air delivery system can be alleviated by shutting down the boiler. However, this is an expensive and unwarranted action and the student will be advised to use an alternative approach.

Heuristic knowledge in RBT is expressed by articulating the steps involved in using detailed operant laws and by explicitly defining the operations performed at the time of a failure. Simply elucidating these operational components and the applicable rules is not sufficient for learning. For this reason, the tutor also provides tools for a student to reason about the complex process. These tools include *graphs* to demonstrate the relationship of process parameters over time, *meters* to measure safety, emissions, efficiency, reliability, and safety, and *interactive dialogues* to tutor the operator about the ongoing process. RBT records every use of these tools and every operation performed by the student.

3. Knowledge of teaching

The second theoretic issue to be addressed in building sophisticated tutoring systems is the representation of teaching knowledge. In addition to being an expert system that solves problems in the domain, a tutoring system must also *teach a student* how to solve those problems in the domain. Because teaching a topic is often more difficult than "knowing" the same topic, a teaching system must be more complex than an expert system. The expert tutoring system will monitor a student's behavior, advise him, respond sensitively to his answers, suggest new activities, and anticipate future actions based on inferences about his current activities.

Building such *teaching* knowledge is critical to development of the tutor. The machine should know how to *ask* the right questions and how to *focus* on the appropriate issue. The system should act as a partner, not as a disinterested, uncommitted, or uncooperative speaker. Effective communication with a student does *not* mean natural language processing (this has been achieved to some degree in systems such as WHY (Stevens *et al.* 1982) and SOPHIE (Brown *et al.* 1982). Rather, effective communication requires looking beyond the spoken words and determining what the tutor and student *should* be communicating about (Collins *et al.* 1975). This problem becomes acute when the student organizes and talks about knowledge in a way that is different from the way the expert organizes and presents it.

Few systems have been effective in the *way* they communicate with the student. GUIDON (Clancey 1982) is an exception that carries on a flexible dialogue with the student based on inferences made about his knowledge. It selects among alternative dialogues based on inferences about the student's previous interactions and inferences about his current information. GUIDON can switch its discussion to any topic listed on an AND/OR graph, representing the rules of the expert system, and can respond to a student's hypothesis using a variety of techniques, one of which is "entrapment," which forces the student to make a choice leading to incorrect conclusions, thereby revealing some aspect of his (mis)understanding.

In this section we describe our experience in building teaching knowledge into two distinct systems, the Recovery Boiler Tutor and the Meno-tutor. The latter system extended our exploration of tutoring issues into the arena of discourse management and understanding. As a precursor to discussion of these tutors, we briefly describe the student model and the role discourse plays in prescribing teaching knowledge.

3.1. Knowledge of teaching in RBT

The student model is an essential component within the knowledge of teaching. It contains the system's knowledge of the student and must be defined in sufficient detail before the system is built to be able to be dynamically updated while the system runs. The student model should not be a simple subset of domain knowledge; it should contain common errors and misconceptions specific to the domain as compiled by domain experts, teachers, and cognitive scientists.

In the Recovery Boiler Tutor, the student model records actions carried out by the student in solving the emergency or operating problem. It recognizes correct as well as incorrect actions and identifies each as relevant, relevant but not optimal, or irrelevant. In RBT, the tutor compares the student's actions with those specified by the knowledge base and uses a simplified differential model to recognize and comment about the difference between the two.

Remediation in RBT is designed to guide without leading. For example, if a partial blackout has been simulated, the black liquor solids are less than 58%, and the operator adjusts the primary air pressure, the tutor might interrupt with a message such as

> "Primary air pressure is one factor that might contribute to blackout, but there is another more crucial factor — try again."

or

> "You have overlooked a major contributing factor to blackouts."

Teaching knowledge in RBT contains the decision logic and rules to guide the tutor's intervention in the operator's actions.

In designing the instructional strategy of the tutor, the intent was to "subordinate teaching to learning" and to allow the student to experiment while developing his own criteria about boiler emergencies. Thus, the tutor guides the student, but does not provide a solution as long as the student's performance appears to be moving closer to a precise goal.

Represented as if/then rules based on a specific emergency and a specific student action, the instructional rules are designed to verify that the student has "asked" the right questions and made the correct inferences about the saliency of his data. Special precautionary messages are added to the most specific tutor responses to alert an operator when a full-scale disaster is imminent. Responses are divided into three categories:

Redirect student: "Have you considered the rate of increase of O_2?"
"If what you suggest is true, then how would you explain the low emissions reading?"
Synthesize data: "Both O_2 and TRS have abnormal trends."
"Did you notice the relation between steam flow and liquor flow?"
Confirm action: "Yes, it looks like rodding the ports worked this time."

The tutor selects a response from each category in an attempt to address the operator's action, his presumed ability to solve the problem, and the need to encourage him to continue to generate hypotheses. Evidence from other problem-solving domains, such as medicine (Barrows and Tamblyn 1980), suggests that students generate multiple (usually 3–5) hypotheses rapidly and make correct diagnoses with only two thirds of the available data.[7]

The RBT was designed to be a partner and cosolver of problems with the operator, who is encouraged to recognize the effect (or lack of same) of his hypotheses and to experiment with multiple explanations of an emergency. No penalty is exacted for slow response or for long periods of trial-and-error problem solving. This is because *learning* requires more exploration and more time than does performance of known activities. In the real world the penalty for slow responses and incorrect action is clear and often painful. This approach is distinct from that of Anderson *et al.* (1985) and Reiser *et al.* (1985), whose geometry and Lisp tutors immediately acknowledge incorrect student answers and provide hints. These authors argue that erroneous solution paths in geometry and Lisp are often so ambiguous and delayed that the source of the error might not be recognized for a long time, if at all, and then it might be forgotten. They argue that immediate computer tutor feedback is needed to avoid fruitless effort.

The trainee in an industrial setting must learn to evaluate his own performance from its effect on the industrial process. He/she should learn to trust the process to provide as much feedback as possible. In RBT we provide this feedback through animated simulations, trend analyses, and "real-time" dynamically updated meters. The textual dialogue from the tutor provides added assurance that the operator has extracted as much information as possible from the data and it establishes a mechanism to redirect him if he has not (Burton and Brown 1982; Goldstein 1982).

[7]Medical students have been found to ask 60% of their questions while searching for new data and obtain 75% of their significant information with in the first 10 min after a problem is stated (Barrows and Tamblyn 1980).

3.2. *Knowledge of teaching in Meno-tutor*

Meno-tutor places more machine intelligence in service to the choice among tutoring strategies. It reasons about the *way* the system communicates with the student and the *topics* that it chooses based upon a model of the student's goals, the domain complexity, and the current discourse history (Woolf and McDonald 1984*a,b*); Woolf 1984). Meno-tutor uses a student model, an annotated domain, and a representation of tutorial planning to custom-tailor its response to the student, both in content and form.

Meno-tutor was able to respond in two domains, reasoning about rainfall and programming loops in Pascal. In the rainfall domain, the student model was based on cognitive research on student misconceptions about rainfall (Stevens *et al.* 1982) and in the domain of Pascal we used groups of programming errors developed empirically (Bonar 1982*a,b*; Johnson and Soloway 1985). Extensive testing and videotaped interviews of correct and incorrect programming strategies yielded high-level procedural plans that are supposedly used by experts to transform problem descriptions into programs.

A major thrust of Meno-tutor research has been to develop the control and data structures needed to plan responsive discourse such as that observed in human tutoring. Meno-tutor (Woolf 1984) is a "generic" tutor, in that it is not committed by design to a single tutoring approach or tutoring domain. Rather, it provides a general framework within which tutoring rules can be defined and tested. Its knowledge of the two domains, in fact, is shallow.

We contrast our work with older tutoring and discourse systems (Brown *et al.* 1982; Burton and Brown 1982; Mann *et al.* 1977; McKeown 1980) that were "retrieval-oriented." While we placed emphasis on choosing among alternative responses that guide the learner based on what the tutor knows about him, other systems have placed emphasis on retrieving a correct answer. Such systems sought to produce a correct answer independent of the user's knowledge or current history. More recent interface and tutoring systems (Finin 1983, Wilensky 1982; Clancey 1982) have begun to tailor their responses to the user and to discourse context.

As an example of discourses produced by the Meno-tutor, we present Fig. 7. These discourses were originally produced in human-to-human tutoring situations (Stevens *et al.* 1982). Meno-tutor reproduced the actual text and high-level transitions in these and twelve other discourses that were used to "reverse engineer" the mechanism of the tutor. That is, we analyzed the common transitions and speech patterns used in these discourses and then defined the structures and knowledge necessary for a machine tutor to have a similar model of the student and to make the same transitions. For example, in the first of the two discourses, the system recognized that the student made two wrong answers and it inferred that his/her knowledge was limited.[8] It then judged that the question answer approach, which had been used until then, was ineffective and should be changed and that a new topic, the "Japan Current," should be discussed because it supplies a dominant influence in the region's climate. The system decided to supply the additional data in a descriptive, rather than an interrogative

[8]It's not that those answers were simply "wrong," rather that they reflect reasonable default assumptions about the weather in "northern states." An attempt to probe the student's default assumptions is made in the second discourse.

Tutor: Do you know what the climate is like in Washington and Oregon?
Student: Is it cold?
Tutor: No, it is rather mild. Can you guess about the rainfall there?
Student: Normal, I guess.
Tutor: Well, the Japan current, which starts in south eastern Pacific, goes along the coast of Japan and across the North Pacific ending up off the coast of Washington and Oregon. How do you think that current affects the climate there?
Student: It's probably rainy.
Tutor: It is indeed very wet, there are rain forests in both Washington and Oregon. What does that tell you about the temperature of the Japan current?
Student: It's warm.

Tutor: Do you think it rains much in Oregon?
Student: No.
Tutor: Why do you think it doesn't rain much in Oregon?
Student: Because the neighboring states are dry.
Tutor: The neighboring states are dry, but, you cannot generalize from that. In fact, it does rain a lot in Oregon. Can you guess what causes the rain there?
Student: Well, there are high mountains in the region and the states border the Pacific Ocean.
Tutor: Yes. The Pacific borders Oregon. How do you think it is involved in the heavy rainfall there?
Student: Ocean air rises and cools upon reaching the land.
Tutor: That is right.

FIG. 7. Two discourses produced by Meno-tutor.

style, because the student seemed confused and might profit from the addition of supplemental data.[9]

The first discourse was generated by Meno-tutor in such a way that all of the decisions, topics, and tutoring styles available to the system were explicit and modifiable by the authors. Modifying discourse decisions allowed us to generate additional discourses moving beyond the "reverse-engineering" of this first discourse. The "tutoring space" defined by our apparatus allowed us to vary the domain and the particulars of the rules. For example, the second discourse in Fig. 7 was based on the same domain as the first, but was done in an alternative tutoring style, brought about by modifying the "meta-rules" that govern whether the tutor first explores the student's frontier or probes his/her misconceptions immediately as soon as the first mistake is observed.

Two meta-rules (see Sect. 3.3) were modified to achieve this second discourse. As explained in the next section, meta-rule can move the tutor to change its tutoring strategy. In the first discourse, the meta-rules in question were used conservatively such that probing a student's misconceptions was implemented only after several topics were completely discussed and the tutor had some confidence in the student's knowledge or lack of same. In this discourse, however, the two meta-rules were triggered after a single incorrect answer, thus shifting the focus of the discourse abruptly to a discussion of misconceptions at the beginning of the discourse.

The modified rules caused the tutor to question the student

[9]Meno-tutor has been developed without a full-scale natural language understander or generator. The conceptual equivalent of a student's input is fed by hand to the tutor (i.e., what would have been the output of a natural language comprehension system) and the output is produced by standard incremental replacement techniques. We have not yet worked with MUMBLE (McDonald 1983), our surface language generator, because we haven't yet invested in building a large enough knowledge base to make the linkup useful.

```
PROGRAM LESSON1(INPUT, OUTPUT);
  VAR
    SUM,GRADES,STUDENTS:INTEGER;
    MEDIAN:REAL;
  BEGIN
    SUM:=0;
    STUDENTS:=0;
    READ(GRADES);
    WHILE GRADES>0 DO
      BEGIN
      SUM:=SUM + GRADES;
      STUDENTS:=STUDENTS+1;
      GRADES:=GRADES+1;
                  → should be READ(GRADES);
      END;
    MEDIAN:=SUM/STUDENTS;
    WRITELN
    ('THE MEDIAN GRADE IS', MEDIAN:8:3)
  END
```

FIG. 8. A student Pascal program.

about misconceptions. In the first discourse, meta-rules were triggered after all topics were discussed, either all questions had been answered correctly or the student was provided with remediation by the tutor. In the second discourse, however, the rules were modified to eliminate that requirement, with the result that both rules were triggered and the discussion quickly centered on the students misconceptions.

In addition to producing a variety of tutoring styles by changing the meta-rules, we explored the tutoring space available to Meno-tutor by substituting a new domain knowledge base in place of the knowledge about rainfall. Using the same teaching mechanism and a knowledge base about elementary PASCAL looping constructs,[10] we demonstrated the power of isolating reasoning about tutoring strategies from reasoning about the knowledge to be tutored. One reason for this was to see if the tutoring component could be interfaced with a different expert knowledge base and a different language generator in order to teach a new subject and even "speak" in a new language. If our modularization was effective (we speculate), we could combine a Pascal knowledge base and, say, a Chinese language generator with the tutoring component, and the resulting system could interrogate a student in Chinese and teach him about programming in Pascal. The difference in domain and language realization should force no changes in the tutoring component, though of course it might be quite inappropriate in China to use the same mix and structure of tutoring strategies as in the English language version of the system.

The program depicted in Fig. 8 was actually submitted by a novice programmer and the discourse shown in Fig. 9 was actually generated by the original Meno-tutor with the indicated changes to the expert knowledge base.

The changes required to produce each discourse are described in Woolf (1984). Though the number of discourses

[10]Meno-tutor was originally developed as part of a larger research effort directed at building an on-line run-time support system for novice Pascal users (Soloway *et al.* 1981; Bonar 1982*b*; Bonar and Soloway 1985). As a part of this effort, a Bug Finder was developed that detected run-time semantic errors in novice Pascal programs (Bonar 1982*a,b*) and passed this information on to Meno-tutor. The Bug Finder could identify the type of error and the line numbers of related variables. It was used for three semesters on classes of several hundred students at the University of Massachusetts.

Discourse Produced Using a PASCAL Data Base

Tutor: Do you know that GRADES, in line 9, is a control variable for the WHILE loop?
Student: Yes
Tutor: Does that mean that GRADES should be
(a) Initiated before the loop?
(b) Added into SUM within the loop?
(c) Modified within the loop?
(d) Both (a) and (c)
Student: C
Tutor: You seem to understand about the control variable of the WHILE loop. Now let's talk about some other command statements.
What do you think READ (GRADES) in line 8 does with the value of GRADES?
(a) Reads all values of GRADES at one time.
(b) Reads in a single value of GRADES.
(c) Assigns several new input values to GRADES.
Student: A
Tutor: Then do you think that GRADES := GRADES + 1 updates the next of those values?
Student: Yes
Tutor: There is your misconception. Adding one to a variable is a common way to step through a range of values. But it cannot apply here because your READ instruction in line 8 retrieves only a single value. You should replace line 13 with a second call to READ (GRADES).

FIG. 9. Meno-tutor discussing errors from the program of Fig. 4.

produced is still small, the fact that our architecture allowed us to produce varied, but still quite reasonable, discourse as we changed the particulars of just a few rules, substantiates the overall effectiveness of our design.

3.3. The discourse management network

The primary mechanism used by Meno-tutor to customize discourse to the individual student was the discourse management network (DMN). Meno-tutor separated the production of tutorial discourse into two distinct components: the tutoring component which contains the DMN and the surface language generator. The tutoring component made decisions about what discourse transitions to make and what information to convey or query; the surface language generator took conceptual specifications from the tutoring component and produced the natural language output. These two components interface at the third or tactical level of the DMN as described below. The knowledge base for the tutor was a KL-ONE network supplied by the author and annotated with pedagogical information about the relative importance of each topic in the domain.

The tutoring component is best described as a set of decision units organized into three planning levels that successively refine the actions of the tutor (Fig. 10). The refinement at each level maintained the constraints dictated by the previous level and further elaborated the possibilities for the system's actions. At the highest level, the discourse is constrained to a specific tutoring *pedagogy* that determines, for example, how often the system will interrupt the student or how often it will probe him about misconceptions. At this level a choice is made between approaches that would diagnose the student's knowledge (*tutor*) or introduce a new topic (*introduce*). At the second level, the pedagogy is refined into a *strategy*, specifying the approach to be used. The choice here might be between exploring the student's competence by questioning him, or by describing the facts of the topic without requesting any response. At the lowest level, a *tactic* is selected to implement the strategy. For example, if the strategy involves questioning the student, the system can choose from half-a-dozen alternatives, e.g., it can question the student about a specific topic, the dependency between topics, or the role of a subtopic. Again, after the student has given his answers, the system can choose from among eight ways to respond, e.g., it can correct the student, elaborate on his answer, or alternatively, barely acknowledge his answer.

The tutoring component presently contains 40 states, each organized as a LISP structure with slots for functions that are run when the state is evaluated. The slots define such things as the specifications of the text to be uttered, the next state to go to, or how to update the student and discourse model. The DMN is structured like an augmented transition network (ATN); it is traversed by an iterative routine that stays within a predetermined space of paths from node to node.

The key point about this control structure is that its paths are *not fixed; each default path can be preempted at any time by a* "meta-rule" that moves Meno-tutor onto a new path, which is ostensibly more in keeping with student history or discourse history. The action of the meta-rule corresponds functionally to the high-level transitions observed in human tutoring. Figure 11 represents the action of two meta-rules, one each at the strategic and tactical level. The ubiquity of the meta-rules—the fact that virtually any transition between tutoring states (nodes) may potentially be preempted—represents an important deviation from the standard control mechanism of an ATN. Formally, the behavior of Meno-tutor could be represented within the definition of an ATN; however, the need to include arcs for *every* meta-rule as part of the arc set of every state would miss the point of our design.

The system presently contains 20 meta-rules; most originate from more than one state and move the tutor to a single, new state. The preconditions of the meta-rules determine when it is time to move off the default path and into a new tutoring state. Meta-rules examine such data structures as the student model (e.g., Does the student know a given topic?), the discourse model (e.g., Have enough questions been asked on a given topic to assess whether the student knows it?), and the domain model (e.g., Do related topics exist?). Two meta-rules are described in an informal notation Fig. 12.

3.4. Summary of teaching knowledge

This brief description of teaching strategies in RBT and discourse management in Meno-tutor has illustrated the kind of complex teaching knowledge a tutoring system must have to make inferences about how to teach in addition to the system's knowledge about the domain. The two systems provide a view of the tutoring space available to an intelligent system and demonstrate how much knowledge is needed to plan and generate responsive tutoring discourse in an intelligent tutor. The key point about the teaching strategies and discourse management is the need for flexibility; strategies must be multiple and linked to complex models of the student and the manager must receive its motivation, justification, and guidance from inferences made about the student and the ongoing discourse.

4. Evaluation

Of the systems described above, only RBT has left the laboratory and is now being used for training in more than 40 industrial sites. The tutors have been placed in the control rooms pulp and paper mills throughout the U.S. Formal eval-

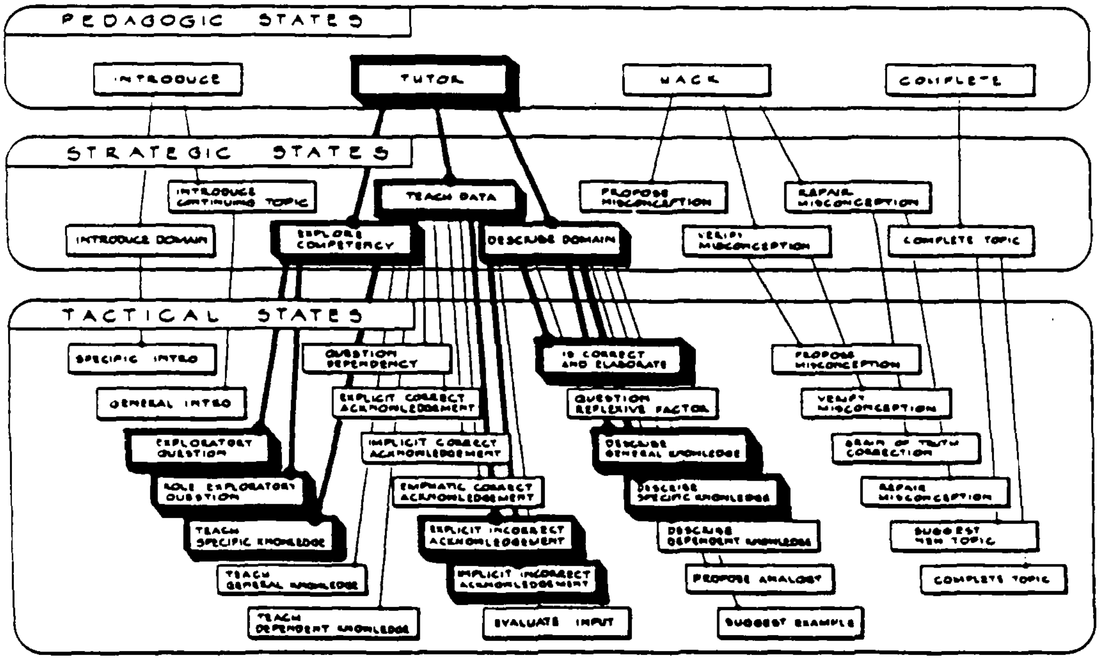


FIG. 10. DMN used by the tutoring component.

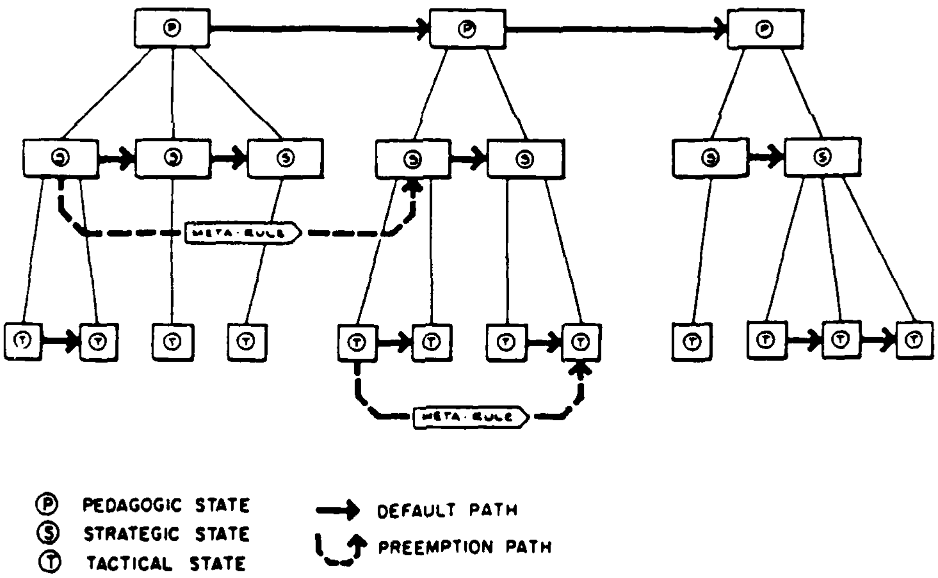


FIG. 11. The action of the meta-rules.

uation from these sites is now being processed. However, informal evaluation suggests that operators use the system and handle it with extreme care. They behave as they might with the actual boiler slowly changing parameters, adjusting meters through small intervals, and checking each action and examining several meter readings before moving on to the next action.

Both experienced and novice operators engage in lively use of the system after about a half hour of introduction. When several operators interact with the tutor, they sometimes trade "war stories," advising each other about rarely seen situations. In this way, experienced operators frequently become partners with novice operators working together to simulate and solve unusual problems. In this kind of setting, the machine becomes the arbitrator or reference point for the discussion. It does not take the role of an authoritarian leader.

5. Conclusions

Several fundamental lessons about building intelligent tutors were learned from developing these two projects. The first and foremost was the need for "in-house" expertise; in the case of RBT, the programmer, project manager, and director of the project were themselves chemical engineers. More than 30 years of theoretical and practical knowledge about boiler design and teaching were incorporated into the system. Development time for this project would have been much longer if these experts had not previously identified the chemi-

S1-EXPLORE — a strategic meta-rule
From: teach-data
To: explore-competency
Description: Moves the tutor to begin a series of shallow questions about a variety of topics.
Activation: The present topic is complete. The tutor has little confidence in its assessment of the student's knowledge.
Behavior: Generates an expository shift from detailed examination of a single topic to a shallow examination of a variety of topics on the threshold of the student's knowledge.

T6-A.IMPLICITLY — a tactical meta-rule
From: explicit-incorrect-acknowledgement
To. implicit-incorrect-acknowledgement
Description: Moves the tutor to utter a brief acknowledgement of an incorrect answer.
Activation: The wrong answer threshold has been reached and the student seems confused.
Behavior: Shifts the discourse from an explicit correction of the student's answer to a response that recognizes but does not dwell on the incorrect answer.

FIG. 12. Informal notation of meta-rules.

cal, physical, and thermodynamic characteristics of the boiler and collected examples of successful teaching activities. This same need for knowledge about the domain and teaching in the domain was evident in building Meno-tutor; it was supplied by prior cognitive studies about the causes of rainfall (Stevens *et al.* 1982)

A second critical lesson learned was the need to clarify and implement the components of a teaching philosophy in concert with development of the expert system itself. For example, in order to manifest a philosophy of subordinating teaching to learning, as realized in RBT, we had to build in several forms of knowledge, including the ability to recognize partially correct as well as irrelevant actions, the ability to custom-tailor the systems's responses to each type of student answer, and the ability to monitor the student while judiciously reasoning about when to interrupt. Each form of knowlege had to be designed in concert with the others; tutoring could not be tacked onto an expert system. The need to limit authoritarian responses from the system and to have it provide as little help as possible meant that the tutor had to be developed as an integral component of the expert system. Even the tutor's ability to remain silent had to be carefully orchestrated. Indeed, the tutor's silence (inactivity) is, in itself, a recognition of the learner's role in the training process and provides an expression of the author's confidence in his progress.

A third, and most surprising, lesson learned from these projects was that tutoring systems can be designed for multiple students. For example, RBT is now used effectively with groups of operators who work together with the computer to solve problems. The computer acts as an adjudicator, not as a dictator. Novice and experienced operators, people who might otherwise not engage in group problem solving, can share experience within a nonevaluative environment; as a group they move ahead to explore unfamiliar situations.

Several issues remain unresolved in our continuing work to improve a computer tutor's ability to respond to the student. For example, we want the system to sort out those skills and concepts that a student has learned from those that he is still trying to learn, and to recognize which techniques have been effective in helping the student. Such knowledge might be built into the student model by way of inferencing.

Certainly, the wealth of knowledge that would enable us to build intelligent tutors is not yet available. In this article we have suggested ways to acquire domain and tutoring knowledge. Further research into sophisticated AI techniques coupled with careful attention to detail and intuitions about teaching and learning are needed if we are to make substantial progress in building machine tutors.

Acknowledgements

The author thanks Jeremy Metz, Bradford Leach, and the A.P.I. Recovery Boiler Committee for their encouragement and support in the building of RBT.

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, NY 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

ANDERSON, J., BOYLE, C., AND YOST, G. 1985. The geometry tutor. Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, CA.

BARROWS, H. S., AND TAMBLYN, R. H. 1980. Problem-based learning: an approach to medical education. Springer Publishing Co., New York, NY.

BONAR, J. 1982*a*. Collecting and analyzing on-line protocols from novice programmers. Behavioral Research Methods and Instrumentation.

——— 1982*b*. Natural problem solving strategies and programming language constructs. Proceedings of the Fourth Annual Conference of the Cognitive Science Society.

BONAR, J., AND SOLOWAY, E. 1985. Models of novice programmer constructs. Human Computer Interactions.

BROWN, J., BURTON, R., AND DEKLEER, J. 1982. Pedagogical natural language, and knowledge engineering techniques in SOPHIE I, II, and III. *In* Intelligent tutoring systems. *Edited by* D. Sleeman and J. S. Brown. Academic Press, Cambridge, MA.

BURTON, R., AND BROWN, J. 1982. An investigation of computer coaching for informal learning activities. *In* Intelligent tutoring systems. *Edited by* D. Sleeman and J. S. Brown. Academic Press, Cambridge, MA.

CLANCEY, W. 1982. Tutoring rules for guiding a case method dialogue. International Journal of Man-Machine Studies, 11. Also *in* Intelligent tutoring systems. *Edited by* D. Sleeman and J. S. Brown. Academic Press, Cambridge, MA.

COLLINS, A., WARNOCK, E., AND PASSAFIUME, J. 1975. Analysis and synthesis of tutorial dialogues. *In* Psychology of learning and motivation. Vol. 9. Academic Press, Inc., New York, NY.

FININ, T. 1983. Providing help and advice in task-oriented systems. Proceedings of the International Joint Conference on Artificial Intelligence, Karlsruhe, W. Germany.

FORBUS, K., AND STEVENS, A. 1981. Using qualitative simulation to generate explanations. Report #4480. Bolt, Beranek and Newman, Inc., Cambridge, MA.

GOLDSTEIN, I. 1982. The genetic graph: a representation for the evolution and procedural knowledge. *In* Intelligent tutoring systems. *Edited by* D. Sleeman and J. S. Brown. Academic Press, Cambridge, MA.

HOLLAN, J., HUTCHINS, E., AND WEITZMAN, L. 1984. STEAMER: an interactive inspectable simulation-based training system. The A.I. Magazine.

JOHNSON L., AND SOLOWAY, E. 1985. Intention-based diagnosis of programming errors. Proceedings of the National Conference on Artificial Intelligence, Austin, TX.

KUHN, T. S. 1970. The structure of scientific revolutions. International Encyclopedia of Unified Science. University of Chicago Press, Chicago, IL.

LARKIN, J. H. 1982. Enriching formal knowledge: a model for learning to solve textbook physics problems. *In* Cognitive skills and their acquisition. *Edited by* J. Anderson. Lawrence Erlbaum Associates, Hillsdage, NJ.

MANN, W., MOORE, J., AND LEVIN, J. 1977. A comprehension model for human dialogue. Proceedings of the International Joint Conference on Artificial Intelligence, Cambridge, MA.

McDONALD, D. 1983. Natural language as a computational problem: an introduction. *In* Computational models of discourse. *Edited by* M. Brady and R. Berwick. MIT Press, Cambridge, MA.

McKEOWN, K. 1980. Generating relevant explanations: Natural language responses to questions about data base structure. Proceedings of the Association of Artificial Intelligence.

REISER, B., ANDERSON, J., AND FARRELL, R. 1985. Dynamic student modelling in an intelligent tutor for Lisp programming. Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, CA.

SOLOWAY, E., BONAR, J., WOOLF, B., BARTH, P., RUBIN, E., AND ERLICH, K. 1981. Cognition and programming: Why your students write those crazy programs. Proceedings of the National Educational Computing Conference, Denton, TX.

STEVENS, A., COLLINS, A., AND GOLDIN, S. 1982. Diagnosing student's misconceptions in causal models. International Journal of Man–Machines Studies, 11. Also *in* Intelligent tutoring systems. *Edited by* D. Sleeman and J. S. Brown. Academic Press, Cambridge, MA.

WILENSKY, R. 1982. Talking to UNIX in English: an overview of UC. Proceedings of the National Conference on Artificial Intelligence, Pittsburgh, PA.

WOOLF, B. 1984. Context-dependent planning in a machine tutor. Ph.D. dissertation, Department of Computer and Information Sciences, University of Massachusetts, Amherst, MA.

WOOLF, B., AND McDONALD, D. 1984*a*. Context-dependent transitions in tutoring discourse. Proceedings of the National Conference on Artificial Intelligence, Austin, TX.

——— 1984*b*. Design issues in building a computer tutor. IEEE Computer, **17**(9): 61–73.

WOOLF, B., BLEGEN, D., VERLOOP, A., AND JENSEN, J. 1986. Tutoring a complex industrial process. Proceeding of the National Conference on Artificial Intelligence, Philadelphia, PA.

SPANDEX: An Approach Towards Exception Handling in an Interactive Planning Framework[1]

Carol A. Broverman
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

November 21, 1988

[1]This work is supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, the Air Force Office of Scientific Research, Bolling Air Force Base, District of Columbia 20332, under contract F30602-85-C-0008, and by a contract with Ing. C. Olivetti & C.

Contents

Chapter 1

Problem Definition

In contemporary work environments, individuals are responsible for carrying out complex and detailed tasks. These tasks often involve multiple interrelated steps as well as communication and cooperation with other employees or agents. For example, a mortgage officer in a bank is responsible for ascertaining relevant information from a prospective home buyer, filling out the necessary forms, and coordinating activity with other agents such as the bank assessor and the head of the mortgage department. A useful application for a hierarchical nonlinear planner [46,51,53] is as an "intelligent assistant" [6,10] for individuals such as the mortgage officer just described. In the modern office, computerized tools are already in place to support an office worker's activities. A planner can be used to plan out the typical procedure to accomplish an office worker's task, automating much of the tool usage and thus reducing the level of complexity to be managed by the human. We note that such a problem-solving environment can be viewed as *cooperative* due to the following features inherent in the environment:

- **The planner is generally not a stand-alone system.** An individual's task often cannot be fully automated. Frequently the user must incrementally supply salient information that will guide the planning process and impose constraints on further development of a plan for a task goal.

- **The user initiates planner activity.** The user invokes the planner for assistance as an automated tool when possible. The user and planner are *cooperating* in an attempt to achieve a common goal, namely the achievement of the goal of the task (see Figure 1.1).

- **Planning is interactive.** The planner relies upon the user to make control decisions as well as to provide missing information necessary to continue the planning process.

- **Planning and execution are interleaved.** A common mode of control in a hierarchical planning system is typified by the *expand-criticize* loop used by NOAH

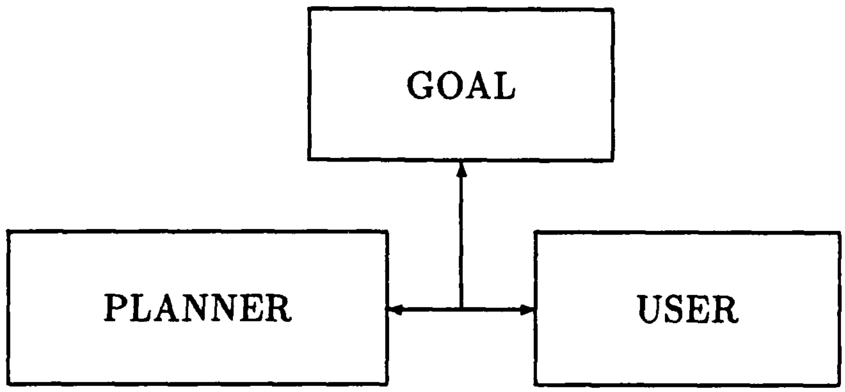


Figure 1.1: A cooperative planner

[46]. In that and other similar planners, a complete level of the plan is expanded and criticized at a time, and all execution takes place only after the plan is completely expanded. In an interactive environment, such as an office, often the planner must wait for the user to execute an information-providing action before it can continue planning. Primitive actions by the user may constitute the execution of steps in the plan. In addition, the planner may execute any primitive actions in the evolving plan whose preconditions are satisfied; these actions may provide information necessary in order to predict what the user should do next towards accomplishing the task. Therefore, the paradigm for planner control in a cooperative planner involves interleaving planning actions with execution, and is better characterized by a ***pick-(execute or expand)*** [32] cycle than ***expand-criticize***.

The execution of a developed plan must be monitored for success. It is important to recognize disturbances to the context upon which the success of the plan depends. For example, if the T-bill rate changes while a mortgage is being processed, the mortgage officer will have to recalculate monthly payment requirements and check them against the applicant's salary. In addition, the mortgage officer or buyer may initiate actions which are not consistent with the system's view of the current task. For example, if the mortgage officer phones the buyer with some information instead of mailing it out, it should be recognized that the mortgage officer may have chosen an alternative route for communicating required information. As just illustrated, often an unanticipated action taken by an agent executing a task is relevant to the plan in some way, and should be interpreted accordingly. Realistically, exceptions and interruptions during the execution of a plan are common occurrences, and an "intelligent assistant" should react to new

information as it becomes available during plan construction and execution.

Current approaches to exception handling in planning have focused on "reactionary" tactics whose primary purpose is to restore the plan to a consistent state. While general replanning techniques are helpful for "recovering" from a problem introduced into a plan, these methods do not attempt to investigate if there are underlying reasons for the exception that may explain how it could be incorporated into the current plan. A general replanning approach is insufficient for an interactive environment since it does not consider the role of the user in generating a possibly valid exceptional action. In this proposal, we present an approach towards an intelligent handling of the types of exceptional occurrences that arise in an interactive planning framework. The overall goal of our approach is to develop a planning architecture which is robust when encountering unanticipated contingencies. In other words, our planning system will be designed to "reason" about exceptional occurrences as they arise, and to include them as part of an evolving plan whenever possible. In addition, we hope to show that future system performance can benefit through exception handling. Specifically, the intelligent handling of an exception can result in the acquisition of new knowledge about performing domain tasks.

The remainder of this proposal is broken down as follows: The notion of an *exception* which can arise in a cooperative problem-solving framework is discussed in chapter 2, and a *taxonomy of exceptions* which are possible is presented. Chapter 3 reviews the work which has been done in the area of exception handling in planning, and discusses the limitations of current approaches in handling the class of exceptions that arise in a cooperative planning framework. In chapter 4, we propose a new approach which attempts to establish the possible roles of an exception within the current plan. This approach will be implemented in a system called SPANDEX[1]. Chapter 4 also includes a comparison on the SPANDEX approach with the current *explanation-based learning* paradigm. The design of the component which reasons about the role of the exception in an evolving plan is outlined further in chapter 4, illustrated by some of the algorithms which will guide the reasoning process. A detailed example taken from the domain of real estate is presented in chapter 5, which demonstrates how two specific exceptions would be handled by SPANDEX. We conclude this proposal with a review of our research goals and a outline of the implementation of this research.

[1]System for Planning AND EXception handling.

Chapter 2

Exceptions

No matter how carefully a plan in conceived, it seems that something frequently goes wrong during its execution, or an unexpected contingency arises which throws the plan awry. Human planners do not always anticipate correctly. People change their minds, or opportunistically revise the plan mid-execution. People also tend to attempt short-cuts while performing a task, or vary their usual methods. In addition, human beings are prone to error or misjudgement. *Consequently, the class of exceptional occurrences*[1] that arise in the cooperative planning framework described is diverse and not sufficiently represented by a set of arbitrary predicates that simply indicate world state changes, as is done by some current systems for replanning [54].

Cognitive and empirical studies have suggested various categorizations of human errors that occur during the execution of a plan [27,44,43]. Upon further analysis, some of these so-called "erroneous" actions can be "declassified" as errors and recognized as purposeful actions which actually are consistent with the plan. It is useful to distinguish between the manner in which an error is manifested and the actual motivation behind an erroneous action. Hollnagel refers to this dichotomy as the phenotype (the way an action appears and is detected) and the genotype (the cognitive basis or cause) of an "action not as planned" [27]. In addition to these two dimensions of unanticipated actions, another aspect to examine is the impact that the unanticipated action has on the current plan. We consider all three characteristics of exceptions in the taxonomy and discussion that follows.

2.1 Taxonomy

One distinction that can be made among exceptions is between "failures" and "surprises" [38]. A *failure* corresponds to the most typical type of planning exception in which new information introduces a problem in the plan, preventing its success. A *surprise* is a more

[1]Throughout the remainder of this paper, exceptional occurrences will often be referred to as *exceptions*.

positive event, denoting the provision of information which may allow a shortening or simplification of the plan. Beyond this simple classification of "harmful" versus "helpful" exceptions, we distinguish between *unaccountable exceptions* and *accountable exceptions*. *Unaccountable exceptions* correspond to unexplained changes in world state brought about by the actions of *unknown agents*[2]. This is the class of exceptions typically addressed by existing replanning systems [54,26]. For example, a system that is planning a travel itinerary may have to contend with the effects of an earthquake which has forced the cancellation of a scheduled train. The procedure which attempts to revise a traveller's travel plan in light of this unexpected event would not (and should not) attempt to establish an unknown agent's motivation behind an exception.

In an interactive planning system, exceptions can also be generated by the actions of known agents such as a user. In particular, a user may perform an action which is inconsistent with system predictions. An unanticipated action *may* be an error. Our belief, however, is that more commonly an unanticipated user action represents an action which is semantically valid in the current plan context. This belief is based on the assumption that *users behave purposefully* when performing an action; their behavior is not simply random. In addition, a user is assumed to be acting towards the same common goal as the planner, and therefore a cooperative relationship between the system and user can be assumed, whereas no such assumption could be made when analyzing the action of an unknown agent[3].

A plan action can occur which is not predicted, but nonetheless valid. This is possible for a number of reasons. The original activity description provided by the system designer may simply have been incomplete. It is also possible that the system was originally provided with an incorrect activity description. Another reason why such actions may not have been predicted originally is that information about the task may be distributed throughout the domain model but is not represented at a level which is readily accessible to the planner. In other words, it may be the case that the algorithms guiding plan expansion do not "notice" semantic links between different action or object descriptions, which if analyzed might provide definitions of alternative methods for accomplishing a given task. The plan expansion algorithms are not designed to take all semantic relationships into account during expansion is because the combinatorics of such a computation at every goal node expansion would be prohibitive. The expansion algorithms are meant to create plans which represent the *typical* methods for accomplishing a task.

We have defined the categories of *accountable exceptions* which can arise in a cooperative planning framework, using a nonlinear hierarchical planner similar to NOAH and

[2]An unknown agent is an agent for which the system has no model, and therefore is unable to reason about that agent's behavior, in contrast to *known agents* who are agents (such as users) that are represented within the system.

[3]Note that unlike unaccountable exceptions, exceptions generated by known agents can be retracted, since users can revise their actions during interaction with the system.

NONLIN [46,51]. Since the behavior of such a system is interactive, user actions are interpreted within the context of two dynamic structures: a plan network which has been partially expanded by the planner, and an *expected actions list* which contains anticipated user actions. The types of exceptions defined by this behavioral perspective are as follows[4]:

1. *Out-of-order action.* The user performs an action that should occur later in the plan. In other words, the user may have skipped one or more steps in the plan.

2. *Repeated action.* The user performs an action that has already been performed in the plan, and is not expected to occur again.

3. *Action not in plan.* The user performs an action which is not expected at all in the plan.

4. *Expected action, wrong parameter.* One or more parameters of an expected action are incorrect. The incorrect parameter may result in two different types of constraint violation:

   (a) *Activity constraint violation.* The incorrect parameter violates a constraint posted in the plan. Constraints are of two types:

      i. *Static.* A static constraint in the activity class definition is violated.

      ii. *Dynamic.* A constraint that was posted dynamically on the partial activity description associated with the activity instantiation is violated.

   (b) *Object constraint violation.* The parameter causes a violation in an object associated with the plan. Object constraints are of two types:

      i. *Static.* A static constraint in the object class definition is violated.

      ii. *Dynamic.* A constraint that was posted dynamically on the partial object description associated with the plan instantiation is violated.

5. *User assertion.* The user provides the description of a state which is to be asserted in the world model. This new state may have an effect on the current plan. A user assertion is modeled as an unexpected action with the assertion as its main effect. It models a state which the user has become aware of, though it is not directly associated with a monitorable action.

In the following section, we discuss the special nature of accountable exceptions in a cooperative planning framework, which is provides much of the incentive for our approach to exception handling presented in the chapter 4.

---

[4]Note that our "behavioral" perspective corresponds to the "phenotypical" view of Hollnagel [27], and the elements of our taxonomy are similar to his "simple phenotypes."

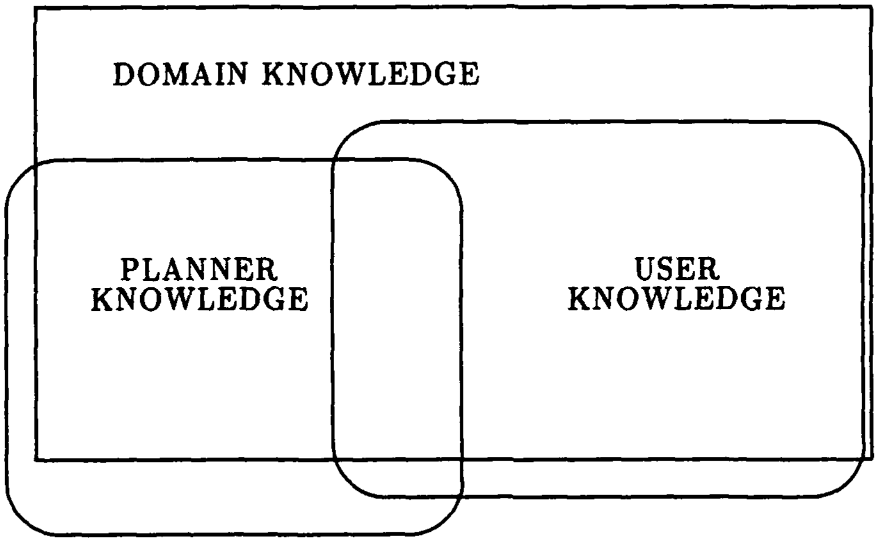


Figure 2.1: Knowledge Distribution in a Cooperative Planning System

2.2 Semantics of exceptions

Accountable exceptions are particularly important in a cooperative planning framework, since they often embody intent on the part of an intelligent agent who is attempting to accomplish a task. Thus, there is good reason to assume that the user had some reason for performing this action, and it is quite likely that the unexpected action is somehow related to the current task. Note that the same can not be said for unaccountable exceptions; they are generated by unknown agents who do not share the system goals.

In general, we assume that users are knowledgeable about the activities for which they are responsible. Thus, in some sense, they are viewed as experts regarding their tasks; and as humans possess knowledge bases about the domain which are generally much less bounded than the initial system domain knowledge (see figure 2.1). Thus, we believe that many accountable exceptions are representative of that portion of domain knowledge which is available to the user, but is unavailable (at least initially) to the planner. In our approach, we are attempting to close the gap between planner and user knowledge through ***exception-driven discovery***. In other words, we would like to be able to focus on user-generated exceptions as a way to gain insight into determining a more complete set of plans for accomplishing a task than that originally known explicitly by the planner.

In addition to additional domain knowledge that the user may have, the user also may know about general planning principles that the planner does not consider. For example, the user may be motivated to take a short-cut while accomplishing a task in order to save

time. Perhaps following an alternative route towards the accomplishment of a plan goal would save the organization some money, another general policy which implicitly guides the behavior of an agent. In general, it would be desirable to acquire and model *policies* of the domain and *motivations* on the part of the various known agents as a further step towards the ability to understand exceptions that arise during plan execution monitoring. Some motivations a user may have when initiating an exceptional action are the following[5]:

- The user may intend a **short cut** of the standard method for achieving the goal of the task.
- The user may intend to **substitute** this action for the actual action anticipated, because of knowledge he may have about alternatives.
- The user may be performing an action which is laying the **groundwork for a later step**.
- The user may know some **information** which may be helpful and eliminate the necessity for some of the actions in the plan.
- The user may intend **partial achievement** of an expected action, which he will need further steps to complete.
- The user may know that the current situation is a **special case**, and the usual constraints on how the task should be done should be relaxed.

Of course, there is always the possibility that the unexpected action initiated by the user is simply an *error*, but our initial belief is that all plausible rationalizations of the action should be considered before we assume that an error has occurred. In the next chapter, we review the work that has been done in the area of error and exception handling, and then continue this proposal by outlining our approach which addresses the special concerns of an interactive planning system.

[5]Note that this classification corresponds to a subset of all possible genotypes [27] of erroneous actions since these are the motivations designated as "contributing" towards the goals of the plan, and do not reflect erroneous judgement or behavior.

Chapter 3

Related work

In this chapter, we review other work which is related to our problem and approach. First, we discuss the achievements and limitations of general replanning approaches. Research in *case-based reasoning* and *explanation-based learning* is also reviewed, since these are two paradigms which are relevant to our view of how exceptions may be reasoned about and learned from.

3.1 General replanning approaches

A number of researchers have recognized the importance of plan execution monitoring and the need to recover from situations where events do not proceed as planned [46,54,50]. Most of these systems are not concerned with unexpected events generated by an intelligent agent such as a user. These systems focus on different aspects of the execution monitoring problem, ranging from the introduction of verification strategies to detect execution results [18] to general sets of replanning techniques which are invoked in response to a detected problem [54]. In preface to a review of these systems, it is important to note that the techniques they provide are primarily relevant to the class of exceptions which we designate as *unaccountable*, since the problem states of concern are not produced by the volitional action of a known agent in the system and therefore usually unexplainable.

In Hayes' robot travelling system [26], replanning is invoked in response to a plan failure, which is defined as the absence of an expected effect of an action, or the recognition of an unexpected event outside of the control of the robot agent. The basic approach taken is to selectively eliminate only the parts of the plan that have been violated by the failure, and to reinvoke the original planning procedure to reconstruct a plan from the earlier point. The major claim of this system is that the plan is "efficiently" reconstructed, and avoids redoing unaffected parts of the original plan.

Nearly ten years later, Wilkin's SIPE system [53,54] continues in this vein, providing

a general replanning capability which is invoked in response to unexpected state changes. Like Hayes' system, one of the most significant aims of the replanning module in SIPE is the capability to recover from a unexpected event during plan execution monitoring, while retaining as much of the original plan as possible. SIPE provides a number of general replanning actions which can be invoked within the definitions of more powerful domain-specific error-recovery instructions. In SIPE, an unexpected situation is represented by an arbitrary state predicate which is entered into the system at an arbitrary point during the execution. The predicate is treated as a new "fact" to be reconciled with the rest of the plan and is inserted into the current plan network as a *mother-nature node*, nomenclature which underscores the arbitrary and external nature of the disturbance to the plan. Problems introduced into the plan by this new state are computed by the *problem recognizer* and the types of problem recognized determine which of the eight general replanning actions are to be invoked. The problems which may be recognized include conditions such as *purpose not achieved, future precondition no longer true*, or *previous phantoms not maintained*. The particular type of problem thus identified triggers the invocation of one or more general replanning actions such as *reinstantiate* (to get a new binding for a variable), or *retry* (to reachieve a goal node that was previously achieved and now violated).

Sacerdoti also recognized the need to anticipate unexpected events caused by outside forces during plan execution monitoring [46]. Upon the detection of a discrepancy in the world model, NOAH would engage the user in a dialogue to verify the parts of the plan leading up to the error in order to pinpoint the source of problem. Once pinpointed, NOAH would plan for new actions to recover from the problem state while maintaining as much of the original plan as possible. In addition the implementation of this type of *error recovery*, Sacerdoti raises the possibility of exceptional states being introduced by an internal agent, such as a system apprentice using a planning system. A human apprentice may provide misinformation about the real world to the system, perhaps necessitating interaction with the user to ensure that the execution of the plan is monitored accurately. While the design and implementation of NOAH did not actually address the issue of user-generated exceptions, it was one of the earliest pieces of work in planning to recognize the need for a more sophisticated handling of this issue.

In all of the above systems, the replanning techniques provided do not attempt to reason about failing conditions or possible serendipitous effects of the exception. These methods simply make use of the explicitly linked plan rationale to detect problems and determine what violated goals need to be reachieved. This type of replanning is "reactionary" tactic used to recover from problems introduced in a plan by events which cannot be reasoned about. Such a general replanning approach should be reserved for handling exceptions generated by unknown agents. The basic replanning approach must be extended to handle the special cases that arise during interactive planning, involving the participation of one or more intelligent agents. Exceptions generated by known agents can be handled by a more

constructive approach which attempts to explain the exceptional behavior. Replanning, as described above, would be inappropriate in these cases, since its primary philosophy is to *counteract* the effects of an exception, which in effect would attempt to achieve goals in a fashion that the user deliberately chose not to pursue. Thus the philosophy of general replanning, as presented here, is fundamentally inconsistent in a "cooperative" planning environment.

3.2 Case-Based Planning

An emerging focus in current planning research is on the reuse of old plans during new plan construction [1,3,21,22,52]. Applying old plans to novel situations is a type of *case-based reasoning* [45,25,29] which is currently a topic of much interest in the artificial intelligence field. Although our work is directly aimed towards the handling of exceptions as they may arise when dealing with an incomplete knowledge base in an interactive system, we share some of the goals and the techniques of these approaches which make use of past plans.

A case-based approach to planning which pays particular attention to failures is exemplified by Hammond's CHEF system [21,22,23,24]. Past plans are indexed by the goals they achieve as well as by failures encountered. In CHEF, a failure is noticed when comparing the results of a plan simulation with expected results. If an expected goal is not among the simulation results, or if an "undesirable" state is reached, a failure has occurred, and a causal explanation of the failure is extracted from the simulation trace. Elements of this *causal explanation* are then used as indices for the retrieval of appropriate plan repair strategies to patch the plan. The memory of the failure is recorded by marking the actual features of the initial situation which are associated with the failure. The major thrust of this work is to show that potential failures in new planning situations can be avoided through anticipation. The anticipation of failures is triggered by the activation of memories of past failures.

Hammond thus relies on a strong causal model of the domain to debug plans which are imperfect and to assign credit to the features at fault. Since organizational procedures have causal information associated with the temporal ordering restrictions, we can employ a similar approach in categorizing problems in the plan network which are caused by the occurrence of an exception. The construction of a causal explanation such as that used by CHEF is a crucial element needed to understand the relevance of a user-generated exception to the goals of the ongoing plan. However, there is an important difference between the approach taken in CHEF and our focus in SPANDEX. CHEF does not do any analysis of the actual operators used in a plan, and does not include plan operators in the abstraction hierarchy used to determine plan similarity. Similarity in CHEF is determined from the comparison of features of plan situations alone, which are abstracted from the goal inputs. We are concerned with understanding relationships between observed and

expected *actions* as well as the actual objects involved in the plan. CHEF pays little attention to the procedural structure of a plan, and the operators of a plan seem quite divorced from the objects which they manipulate. We have taken a more integrated view of plan operators, objects, and agents, and our activity descriptions are quite complex.

Tenenberg's [52] work on the use of abstraction in planning can also be considered to be a case-based approach. This research develops an algorithm to constrain search for a new plan by making use of condensed old plans, in the form of plan graphs. This work demonstrates a recognition of the utility of a generalization hierarchy of both plan operators and domain objects when constructing a new plan. Our work takes a similar tact in making use of taxonomic links as a primary semantic resource to explore when encountering an exceptional action or object. Like Tenenberg, we also see this approach as one which provides savings in search complexity. Whereas Tenenberg constrains the search using an abstract plan graph condensed from an earlier plan, we constrain our search for a "consistent" plan by constraining our search to the knowledge closely related to either the expected action or the exceptional action.

Alterman's planner PLEXUS [1] implements an approach which emphasizes the structure of background knowledge as an important means for altering old plans to fit new situations. Alterman's *adaptive planner* treats failing steps of a plan as representative of the category of activity that should be performed, and thus traverses generalization and specialization links in order to find a step which fits in the new situation. We make similar use of the static library of domain activities and objects, by exploring the relationships between unexpected actions and object parameters and those used in a stereotypical plan for the task. Alterman concentrates on resolving any of four identified types of *situation differences* that can arise in trying to refit an old plan to a new situation: *failing preconditions, failing outcome, differing goals,* and *step-out-of-order*. These classes of situation difference are similar to our behavioral taxonomy of exceptions, although they are in a sense an amalgamation of both exception *class* and *resulting problems*.

Whereas the similarities between our work and Alterman's are numerous, our goals differ. Alterman is attempting to construct a new plan for a known novel situation, while we are in a sense trying to *recognize* an ongoing modification of an existing plan. Also, while we assume a great deal of initial knowledge about the domain we are operating under the premise that our knowledge may be incomplete. Alterman, on the other hand, seems to assume a knowledge-intensive environment which is complete. An additional difference is that in Alterman's work, the planner is responding to new constraints imposed by the "situation," whereas in our environment, the planner must respond to unusual actions by the user and is inherently performing some recognition. In our approach, we would like to take advantage of having intelligent agent(s) "in the loop." Alterman hints at the fact that his system is an interactive one, yet the user seems to be a passive agent and is not engaged in a dialogue to provide guidance during the construction of the new plan.

3.3 Explanation-Based Learning

Much work has been published recently under the rubric of *explanation-based learning*[1]. The goal of the explanation-based learning approach is to construct an *explanation* for a novel concept, and then to generalize the explanation of the specific instance into a more general and operational concept definition. More formally, explanation-based approaches apply a *domain theory* and a *goal concept* to a *single training example* to construct an *explanation* as to how the training example is an instance of the goal concept. Techniques are then applied to transform the specific explanation into a more general specification of the learned concept.

The work being done by DeJong et. al.[13,14,15] is particularly relevant to our work in handling exceptions during plan execution. DeJong's *explanatory schema acquisition* (ESA) is couched in problem-solving terminology rather than being presented from the more typical theorem-proving viewpoint. This orientation is more compatible with our planning perspective, incorporating preconditions, effects, goals, and motivations. The entities used by their systems are actions, states, and objects; similar to our activities, objects, relationships, and states. The general goal of ESA is to construct and generalize an explanation for an observed novel sequence of problem-solving operators. DeJong often refers to his approach as *learning by observation* [15], a term which could also be applied to our approach which "notices" an exceptional action during plan execution monitoring, and attempts to construct an explanation for the role of that action in the plan. In addition, DeJong addresses the need to infer missing steps. This is similar to the determination of activity that potentially may have occurred "off-line," which is a technique used in SPANDEX in response to an *action-out-of-order* exception (see section 4.3.2). However, our work differs from DeJong et. al's in the following ways:

1. Our planning paradigm is an interactive one, so plans are developed and executed in an incremental fashion. Therefore, we do not have a full-fledged detailed sequence of actual plan operators invoked from which to construct an explanation.

2. We are often unable to construct a complete explanation for an exceptional problem-solving strategy taken by the user and must resort to negotiation in order to acquire truly new information to complete the explanation. In other words, we are not assuming a complete knowledge base to start, and in addition to the use of explanation as a method for *restructuring* the knowledge base (DeJong), we also provide for the acquisition of completely new knowledge about the domain.

Consistent with our presupposition, Rajamoney et. al [42] also initially assume an incomplete and incorrect model of the domain. Similarly, they use the observation of an

[1] I am using the term *explanation-based leaning* to subsume other approaches designated as *explanation-based generalization*, *analytic learning* and other similar paradigms.

unexpected effect in the world model as an opportunity to extend and debug the domain knowledge of the system. Their approach is to hypothesize known processes which might have caused the inconsistency, and to use directed experimentation to question the beliefs which fail to support the existence of these processes. This approach differs from the approach taken in SPANDEX in that it addresses only the type of exception which may be generated by an underlying "agent-less" process. In SPANDEX, we take a more behavioral viewpoint, and are interested in handling the cases arising in an interactive planning system where intelligent agents actually carry out much of the execution of the task.

In the next chapter, we propose the architecture of SPANDEX for handling exceptions as they arise in an interactive planning framework. We then show how our approach can be mapped into the explanation-based perspective, and highlight the novel aspects of our approach which are not easily mapped into EBL terms.

Chapter 4

A new approach for handling exceptions

The basic notion guiding our approach is to use the class of a detected exception together with a heuristic determination of user intent, to select among a set strategies which attempt to discover a role for the exception in the current plan, if one exists. Otherwise, the system will attempt to negotiate directly with responsible agents in an effort to require additional explanatory information, or replan if necessary.

A proposed architecture (SPANDEX) designed to accommodate exceptional occurrences is shown in Figure 4.1. Several of the modules are similar to those described in other hierarchical planners, specifically [54]. We have extended a basic planning model to include additional modules to address exception handling. Exceptions are detected by the *execution monitor* and classified by the *exception classifier*. Violations in the plan caused by the introduction of an exception are computed by the *plan critic*. Exceptions generated by unknown agents (generated by *world* in the diagaram) are handled by the *replanner*. The replanning approach we have adopted is similar to that of [54], where one or more of a set of general replanning actions is invoked in response to a particular type of problem introduced into a plan by an exceptional occurrence. For interactive planning, we extend the set of general replanning actions to include the insertion of a new goal into the plan.

The *exception analyst* applies available domain knowledge to construct *explanations*[1] of an exception. Its primary function is to determine the relationships and compatibility of the actual events to the expected actions, goals and parameters. The particular entity relationships investigated by the exception analyst are determined by the type of internal exception. The exception analyst may be triggered by both unaccountable and accountable exceptions, although it is primarily used for accountable exceptions (resulting from the actions of *agents* in the diagram). A more formal view of the role of the exception analyst,

[1] A more precise definition of what constitutes an *explanation* is given in section 4.5.

along with details about algorithms which guide the search process is discussed in section 4.2.

The paradigm of *negotiation* [19] has been used as a model for reaching an agreement among agents on a method for accomplishing a task. We propose to use negotiation for establishing a consensus among agents who are affected by an exception. We distinguish between *effecting* and *affected* agents with regard to the occurrence of an exception. The *effecting* agent is that agent who has caused the exception. An *affected* agent is one whose interests are influenced (either positively or negatively) by the exception. Affected agents are those who are "responsible" for the parts of the plan where problems are detected by the plan critic. An unknown agent can never be an *affected* agent, since the system has no model of an unknown agent's interests or behavior.The *negotiator* determines the set of affected agents and uses the information provided by the exception analyst to complete and choose among explanations as well as to suggest changes to be made to the original plan.

Using information provided by the exception analyst about relationships between actual and expected values, the negotiator initiates an exchange between the effecting agent and the affected agents. The negotiator and plan critic execute in a loop in which the plan critic analyzes changes suggested by the negotiator to detect any problems introduced. This loop is exited when no further problems are detected by the plan critic and all affected agents are satisfied. The negotiator also directs the acquisition of information from the user, if required to complete an explanation constructed by the exception analyst. The output of the negotiation phase is one or more verified explanations, and possibly changes that should be made to the current plan or static plan library.

The *explanation generalizer* produces a generalized form of the verified explanation(s), if possible, using taxonomic information in the knowledge base. This new knowledge about domain activities, along with suggested changes to the knowledge base resulting from negotiation, is passed to the *knowledge base modifier*. Thus, a successful negotiation can result in a system which has "learned," that is, the static domain plans may be augmented with knowledge about the exception and thus the system has an enhanced capability to handle future similar exceptions.

Once an exception has been handled in this fashion, control is returned to the planner to continue plan execution and generation.

4.1 Representation

The general approach just presented obviously requires a rich knowledge base about the domain. Procedural, structural, and causal information are all important aspects of domain knowledge, and should be represented. In SPANDEX, we have chosen a uniform

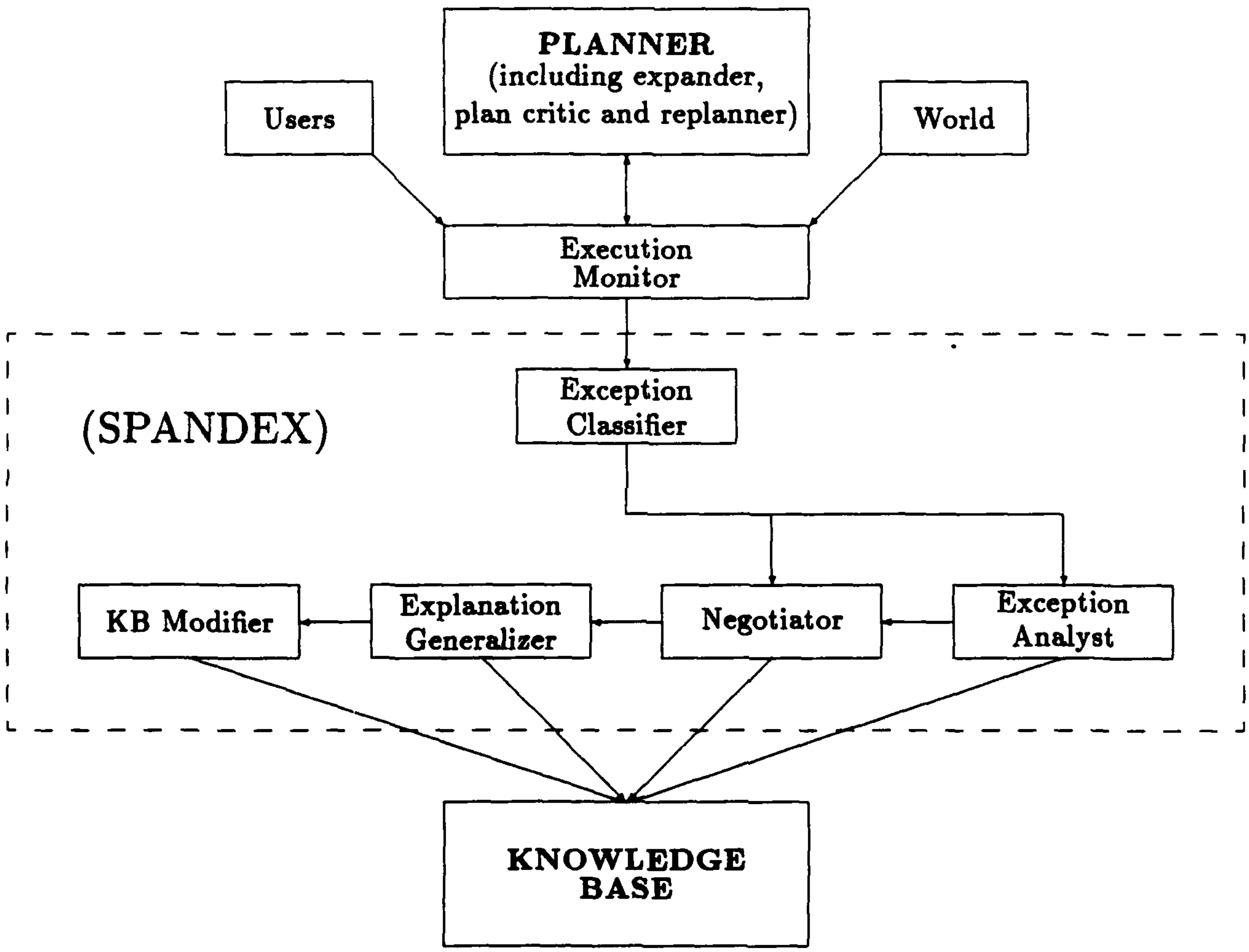


Figure 4.1: Architecture for a cooperative planning system

object-based representation of *activities*, *objects*, *agents and relationships*[2] [6]. An integrated abstraction hierarchy (see Figure 4.3) combined with a powerful constraint language facilitates the representation and use of more sophisticated knowledge about plans, such as the *policies* of McDermott [32]. The reasoning process used by the exception analyst exploits this object-based representation. Similar approaches have been used by Alterman [1] and Tenenberg [52] to represent old plans that are adapted to new situations.

The major features of our representation are *taxonomic knowledge, aggregation, decomposition, resources, plan rationale* and *relationships*. Each of these is defined and illustrated using an example from the domain of house-purchasing, shown in Figures 4.2 and 4.3. Figure 4.2 depicts a partially expanded procedural net fragment which represents the portion of a house-buying task which remains after a house has been selected for purchase. Figure 4.3 shows a portion of the domain knowledge relevant to this task.

Any complex entity can be viewed as a composition of several other objects as well as an aggregation of properties. An abstract activity object which can be decomposed into more detailed substeps has a *steps* property containing a partial ordering of more detailed activity steps. Decomposition of a domain object into other objects is expressed as a set of object types named in a *parts* property. The aggregation of all properties of either an activity or domain object, including decomposition information, constitutes the object definition.

All entities are represented in a type hierarchy, with inheritance along *is-a* links between types and their subtypes. Entities inherit the properties and constraints of their supertypes. For example, a *mortgage-application-form* has various fields which are inherited from the more general *form* object, and obeys the constraint stating that it can be manipulated by an *apply* type of activity (inherited from *application-form*). Activities inherit the preconditions and effects of their supertypes, as well as decomposition information. For example, any apply activity may be decomposed into an activity of type *go-to-place* followed by *fill-out-form*. *Apply-for-mortgage* is a subtype of *apply* and thus inherits and specializes this decomposition. *Apply-for-mortgage* also inherits the effect of *pending(application-form)*.

An activity has an associated set of *effects* which are asserted upon its completion. Effects are represented as predicates on domain objects. The *goal* of the activity is a distinguished main effect and is used for matching during plan expansion. An activity schema also includes a declaration of the types of domain objects it may manipulate. The inverse of this *resources* property is the *manipulated-by* property expressed in domain objects to indicate which types of activities may affect them. The union of an activity schema with the descriptions of associated object types provides a rich semantic representation of the domain, incorporating objects and operators.

[2] In the remainder of the paper, we refer to plan descriptions as *activities* and objects of the domain simply as *objects*.

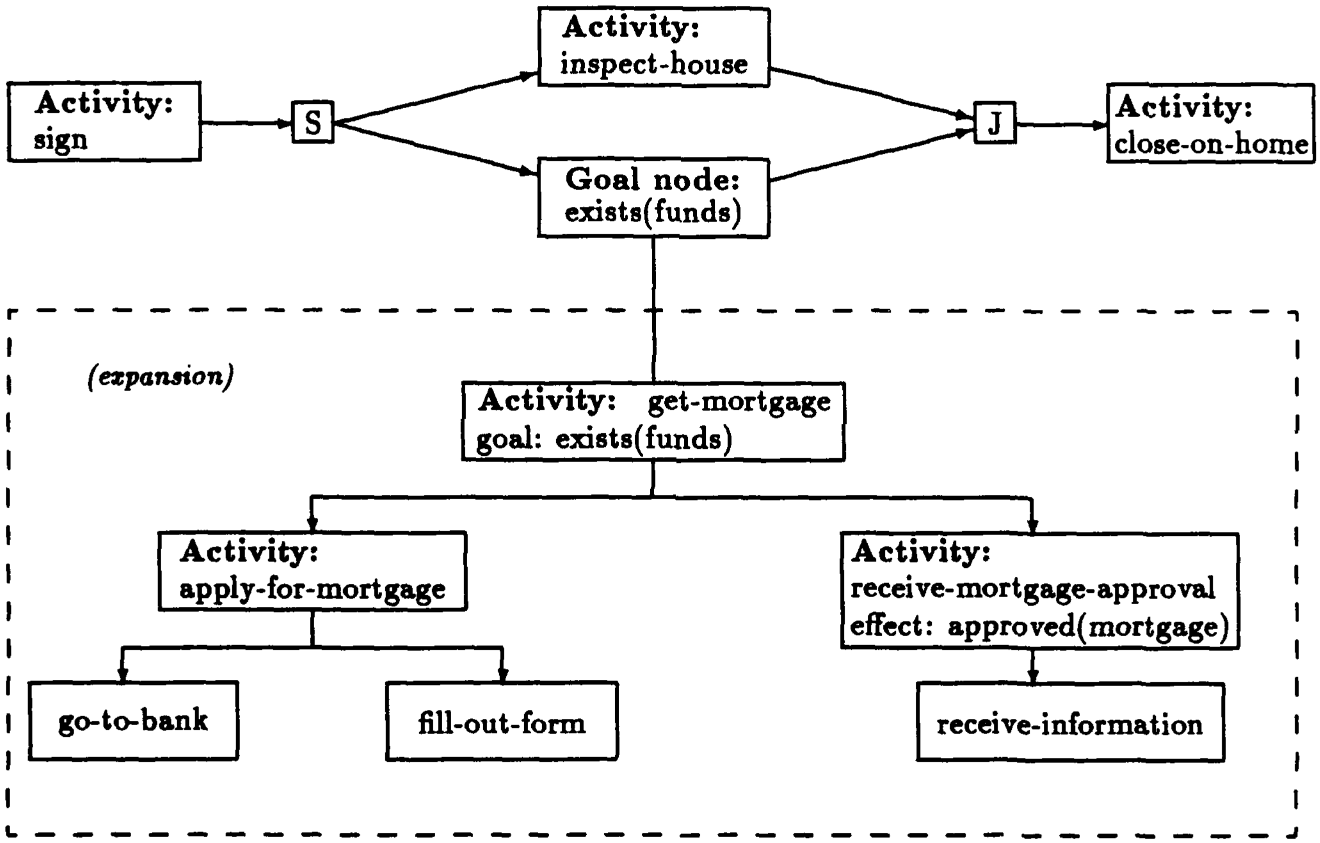


Figure 4.2: Example procedural net fragment

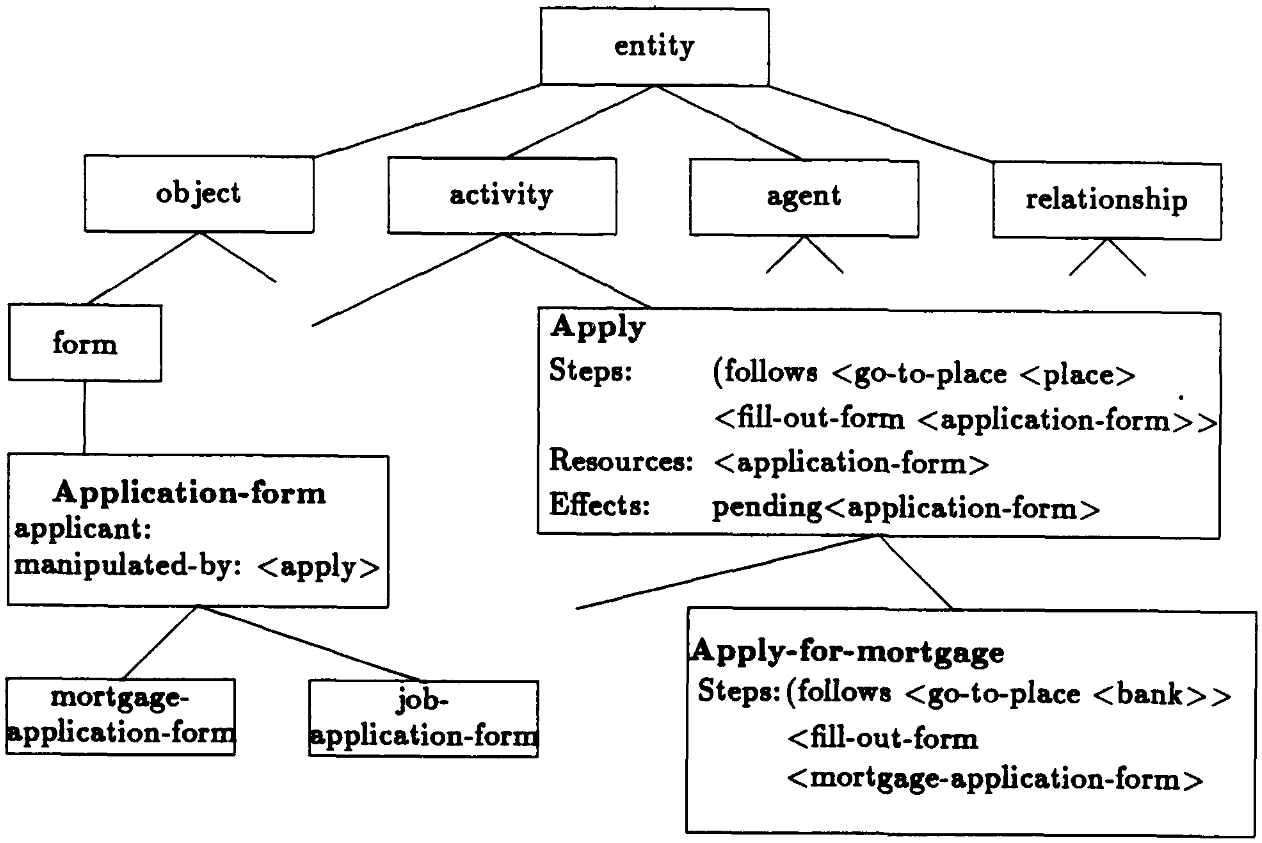


Figure 4.3: Fragment of knowledge base

Causal knowledge is represented by *goal* properties and *purpose* links. *Goals* are of a global nature, in that they relate an activity to a representation of its intent; that is, they state what this activity accomplishes regardless of the context of the current procedural net. *Purpose* links may be placed between two plan substep nodes in both static and dynamic plan representations, to indicate that a substep of a plan produces a state required for the proper execution of a later substep, much like NONLIN's goal structure [51]. The *purpose* links prove to be particularly important in determining whether or not an exception can easily be incorporated into an existing plan.

Arbitrary *relationships* may also exist between domain objects. For example, a *seller* relation may be depicted between an individual and a certain *house*, expressing the fact that someone is selling a particular house. A special type of relationship which may exist between two objects is a *transformation* relation, which contains a procedural attachment for producing the correct instance of one type of object associated with the instance of the second object type. For example, the abstract class object *address* may be related to *telephone-number* through a special transformation specification which indicates that a phone call using a *phone-number* may produce the corresponding *address*.

4.2 Exception analyst

The exception analyst is the component of the SPANDEX system that is responsible for constructing explanations for an exception. It is invoked in the context of a partially expanded and executed plan network, a set of expected actions and goals, an unexpected occurrence, and the current state of the world model. Its task is to produce explanations of the unexpected occurrence within the provided context. There are three classes of explanations which can be produced by the exception analyst:

- **Complete explanations.** In a complete explanation, the relationships verified by the exception analyst are logically sufficient to explain the role of the exception in the plan. Complete explanations are obviously the most desirable explanations and search paths which may result in a logically complete explanation are pursued before all others.

- **Partial explanations.** In this type of explanation, verification is made of certain *leading indicators*[3] among entities pertinent to the expected and unanticipated occurrences, but these leading indicators alone are not logically *sufficient* to completely explain the exception. Additional conditions need to be verified, or missing information must be acquired from agents in order the complete the explanation. Partial explanations constitute explanation skeletons which must be massaged further if no complete acceptable explanations are produced.

[3]These are discussed further in the following section.

- **Null explanations.** A null explanation is essentially a *failed* complete or partial explanation. For example, if a key subclass relationship is searched for and found not to exist, a null explanation can be constructed which consists of this failed test. Null explanations may be fallen back on by the negotiator when all else fails. The assumption of an incomplete database allows for the possibility that an agent may be able to specify the existence of an initially absent key relationship, and thus activate the previously null explanation.

The exception analyst may be formally regarded as providing intelligent search. In other words, given: a) a set of axioms and heuristics which define how planning is performed, and b) explicit and available domain knowledge, a search space is defined for the generation of valid plans. The plans which exist in this constrained space represent stereotypical plans for accomplishing a given task goal. In a knowledge-rich environment, it may be the case that knowledge is scattered throughout the various parts of the knowledge base which would justify other valid plans for the given task. These plans may not have been in the initial plan space because of the constraints imposed by the standard plan generation process. The exception analyst intelligently extends the space searched for valid plans. It does this by looking in the knowledge base for relationships between the unexpected occurrence and the current plan which might justify the role of the exception in the plan. This activity is constrained by the determined exception class and suggested motive. Formally, this amounts to a **relaxation of plan axioms** that were used during the initial plan expansion.

The behavior of the exception analyst is guided by some general principles derived from the type of the exceptional occurrence. A *action-out-of-order* exception, for example, may imply that the user may be attempting a short-cut, while an *action-not-in-plan* exception may be eventually recognized as an intentional substitution of the unanticipated occurrence for the expected step. The exception analyst performs a controlled exploration throughout the knowledge base which is guided by the current state of the procedural network as well as the type of exception which has occurred. If a number of strategies are possible, the least costly is attempted first. In the following section, we present algorithms for handling the various types of exceptions, illustrating (where relevant) with the example scenarios relevant to the example represented in Figures 4.3 and 4.2 in section 4.1.

4.3 Exception analyst algorithms

When an exception occurs, the first task faced by the system is to determine the exception type by invoking the exception classifier. If the user has provided the system with a *user assertion*, the exception classifier recognizes this immediately, and the exception analyst is invoked to treat this unusual occurrence as an *action-not-in-plan*. It is also easily apparent

when an *unexpected parameter* exception occurs, since the user step type will match one of the expected actions or goals, and the parameter mismatch is what triggered the recognition of the exception. *Repeated-step* exceptions are also detected in a straightforward manner.

In the remaining cases, the most difficult detection task for the exception classifier is to designate the unusual user action as one potentially meant as a later step in the plan (*out-of-order*) versus a step not expected in the plan at all. The reason why this task may be difficult is that the plan network is usually only partially expanded (pending information which will be derived from later user actions) and it is not immediately clear what actions will eventually make up the plan at the most primitive level. Therefore, the exception classifier will have to try and fit the exception into a viable extension of the current plan network by performing a search through possible expansions of the incomplete plan network. Note that this is largely a plan recognition task (we are attempting to choose among alternative expansions with incomplete knowledge), and thus is fraught with the associated combinatorial and backtracking considerations.

Once the classification of the exception has been completed by the exception classifier, the exception analyst determines if the exception may somehow contribute to the plan being developed. The exception analyst constructs the set of viable explanations for possible roles of the unusual occurrence. The process by which the exception analyst searches for explanations is guided by the class of exception, which was just determined. In the remainder of this section, we sketch the algorithms which define this search, discussing the exception types on a case by case basis.

4.3.1 Action Not in Plan

If a user action occurs which is not expected anywhere in the plan, the exception analyst attempts to establish whether this unexpected action contributes to the pending task in any way. The fundamental assumption is that the unexpected action is related to some step in the remainder of the plan.

The unexpected action may be related to one of the currently expected steps or to another plan step which is predicted later in the plan expansion. The actual contribution made by this exceptional occurrence can be at an arbitrary level of abstraction and granularity within the task. In other words, an action may take the place of an expected action, satisfy the precondition of a later action, or eliminate the necessity of an entire sequence of pending or future actions[4]. The effects of the actual action are compared with the preconditions, effects, and goals of other nodes within the procedural net. The exception analyst looks for the potential contributions by incrementally a) extending the

[4]This possibility may be actualised by allowing the unexpected action to replace the activity which contains an expected action, since all other steps of that replaced activity would no longer be necessary. Also, note that the idea of a single unanticipated occurrence corresponding to a string of expected steps represents an example of the class of *phenotypes* that Hollnagel refers to as *complex*.

locality of focus (as defined by the level of abstraction) and b) decreasing requirements on the completeness of the explanation. The control of the exception analyst while searching for **complete** explanations is illustrated by the following algorithm:

1. Can the exceptional action be *substituted* for an expected action? If either of the following criteria are met, a substitution should be allowed:

   (a) The goals of the expected and unexpected actions match. The unexpected action achieves the same world state as an expected action.

   (b) The exceptional action is a specialization of an expected action, and thus is sufficient to accomplish at least what the expected action would have accomplished.

   (c) Effects of the exceptional action *exactly* match those of the expected action.

   (d) The intersection of effects of the exceptional and expected actions are exactly those effects of the expected action which have *purpose* links to later plan steps.

2. Does the exceptional action allow a *simplification* of the remainder of the plan?

   (a) If the action can be substituted for a *later* step in the plan (established by the above method), treat the exception as an *out-of-order* action (below) and record the substitution of the matching actions.

   (b) Do any of the effects of the exceptional action match with an unachieved effect which is the *purpose* for a later plan step? If so, a later precondition is satisfied; note that the precondition is now a phantom, but do not modify expectations.

3. Does the unexpected action allow an entire hierarchical wedge to be removed from the plan?

   If the exceptional action results in the satisfaction of a higher-level goal, the steps comprising the expansion of that goal may no longer be necessary. The exception analyst determines the parent node of the expected action. If the goal of this parent node is achieved by the effects of the exceptional action, then the following is done: Check to see if the effects of each of this parent's children (excluding exceptional action itself) are now true. If none of the *un*achieved effects have *purpose* links to steps occurring after the parent node, then a substitution is allowed. The exceptional node is incorporated in the procedural net, and the expected action, its parent and siblings are considered to be achieved.

If none of the above criteria are met, then it is not possible to easily integrate the exceptional action into the existing plan network by replacing part of the plan network

with the representation of the unexpected action. It is still possible, however, that the unexpected action constitutes a partial contribution to a pending plan goal, and partial explanations may be constructed. We are not focusing on this case at the moment, but we initially propose repeating the above procedure, looking for features such as the *leading indicators* outlined in the section on *unexpected parameter* exceptions.

4.3.2 Out-of-order action

If the action is judged to be an out-of-order plan step, there are two possibilities to consider:

1. The original ordering may have been specified as a preference, but there are no strict dependencies between the effects and preconditions of actions. In order to determine if this is the case, the exception analyst must examine the causal structure of the plan. Specifically, if there are no *purpose* links between the actual step and an intervening step which has not been performed, the ordering may be relaxed.

2. The intervening steps between the expected and actual actions are no longer necessary. This may be because the goals of the intervening steps may have been accomplished in some "offline" fashion. A procedure is invoked to attempt to prove (deduce) that the goals of the intervening steps have been accomplished. If this is not successful, control is passed to the negotiator, which involves the user in an attempt to verify the goals of the intermediate steps.

4.3.3 Unexpected parameter exceptions

Unexpected parameter values can cause constraint violations. Since parameter values are usually objects themselves, the exception analyst is invoked to determine what relationships exist between the object provided as the *actual* parameter value and the object which was *expected* as the parameter value.

There are a couple of different cases when parameter discrepancies may arise. Often, an action may have been placed on the expected-actions-list with some of its parameters bound to known values, and other of its parameters unbound, pending information to be derived from a user action. Thus, we may have a case where the user action monitored by the system provides the system with a value for the unbound parameter, but the values provided for the other ("known") parameters are inconsistent. In this case, the exception analyst must compare two *instance* objects, which may or may not be of different types.

A different case arises when an action is predicted to occur, and the parameter types are also predicted, but not the actual values (an information-getting action). The user action may provide a binding for the the parameter(s) in the form of an object instance specification whose type does not match the type predicted. In this case, the exception analyst must determine the compatibility of the two type specifications.

A third situation in which a parameter discrepancy may arise is during an attempt to substitute an unexpected action for an expected action (see the section above on unexpected action exceptions). An unexpected action may be found to be a specialization of an expected action, or otherwise related in a way that would allow the action to be substituted. It may, however, be the case that the parameters do not match. This type of discrepancy may involve the analysis of two type specifications, two instance specifications, or the specification of a type and an instance.

For all of the cases just described, the exception analyst attempts to determine what relationships exist between the expected parameter type (or value) and the unexpected parameter type (or value). Note that the first two of the relationships specified may constitute *complete* explanations in this regard, while the remainder can only be viewed as *leading indicators*, since they represent significant relationships that can form the basis of an explanation, but are not sufficient alone as an explanation. The relationships examined by the exception analyst in this case are the following [5]:

1. The type of the unexpected instance may be a specialization of one of the expected instance types, and may be substituted.

2. If we are comparing two object types, it may be that the unexpected type is a *specialization* of the expected type. This type of discrepancy is easily resolved, since the specialized object has all the properties of the expected object, and thus is sufficient. (It may be the case, however, that this object has additional properties which may be problematic in the plan; the plan critic must check this).

3. The two objects may both be *manipulated-by* activities which belong to a common activity superclass. If so, they probably are utilized in similar fashions.

4. Both may be instances of the same type. It is possible that the two instances may be substituted for one another.

5. The type of the unexpected instance may subsume the type of one of the expected instance types. The system should determine what the properties of the expected parameter type are missing, and determine through negotiation whether these properties are necessary. If not, the more general form of the object should have been specified initially.

6. The unexpected parameter type may *subsume* the expected parameter type. That is, it is not specific enough. The system should determine what the properties are

[5] Note that when comparing *objects* as opposed to *activities*, we do not focus on effects or goal similarities, since these are only relevant for activity descriptions. The specialisation hierarchy and other arbitrary relationships are explored in more detail. Also, this classification of relationships subsume a previously defined classification of similarity offered by Simpson [49].

of the expected parameter type that are missing, and determine through negotiation whether these are necessary properties. If not, the more general form of the object should have been specified initially.

7. The unexpected parameter object may be *part-of* the expected parameter. We could go into negotiation to ask the user for a specification of the other *parts*, or see if the operation (action) can be performed on the part alone.

8. The unexpected parameter object may *contain* the expected parameter type. The exceptional action may have been a "slip," we can derive the expected parameter value from the unexpected one.

9. There may be some other relationship between the unexpected and expected parameter types. The semantics of this relationship (at least the mappings) are available in the system knowledge base; the exception analyst can determine the expected object type from the unexpected object type. The negotiator can then ask the user if this is what he meant.

10. The two objects may be *siblings* in the object hierarchy. If so, the exception analyst constructs the set of features *unique* to the expected object, since the lack of these features in the object actually provided as the parameter value may be problematic. Negotiation must determine if aspects missing or extraneous in the unexpected object are problematic.

11. The discrepancy between the two parameters may result from *differing quantities* of the object type. If so, an excess may or may not be allowable. The semantics associated with the underlying data type are particularly important when handling quantity discrepancies, since commonsense reasoning may be required. For example, if the *go-to-bank* step was supposed to result in withdrawing 50 *dollars*, emerging with 100 may not be problematic, but baking a cake in a 450 *degree* oven when the recipe calls for 350 degrees may have unsatisfactory results.

Once the exception analyst has investigated the above relationships, a set of possible partial explanations has been constructed. Only rarely is an explanation complete enough to assume as correct without further information and verification from the user (perhaps the only case is straight specialization). The negotiatior is handed the set of partial explanations to "discuss" with the user. These explanations may have been ordered by a heuristic rating scheme to represent the likelihood of validity. The information acquired during negotiation establishes whether the exceptional parameter should be allowed.

4.4 Negotiator

The negotiation process is used to verify or complete the explanation of the exceptions encountered during the execution of activities. The primary goal of the negotiator module is to ascertain that the plan network has been returned to a consistent state, and all affected agents have agreed upon any changes that were made during the handling of an exception. By "consistent," we mean that if an exception has occurred, at least one acceptable explanation has been formed and verified, and the outstanding goals of the plan cannot be shown to be unachievable[6]. The activity of the negotiator has three general phases:

1. The agent who caused the exception and the agent who are *affected* by the exception are identified. In the general case, there may be multiple affected agents, but oftentimes the only affected agent is the effecting agent himself.

2. The negotiator moderates an exchange between the affected and effecting agents to establish a consensus about one of the possible explanations. In general, depending on the completeness of the explanations, the nature of this phase of the negotiator is one of two possible types:

   (a) **Verification.** When one or more complete explanations are found, they are presented to the affected agents (after possibly being ranked) to choose and verify the most acceptable explanation. Again, in the case where a single agent is both the effecting and affected agent, this phase defaults to asking that agent to choose among the potential explanations constructed by the exception analyst. This kind of negotiation will lead to a generalized explanation which represents a *reorganization* of existing domain knowledge.

   (b) **Guided acquisition.** When no complete explanations are found, the exception analyst provides the negotiator with a search trace of the examined entities and relationships which failed to support potential explanations of the exception. Using this trace, the negotiator carries out a directed query with the affected agents in an attempt to acquire missing information which could constitute an explanation. This mode of negotiation can result in *new* domain knowledge.

3. The negotiator must determine what changes should be made to the current plan network and/or static activity and object library in response to the handled exception and explanation. Again, all affected agents must agree on the changes to be made.

[6]Note that this is very different from requiring that the outstanding goals be shown to be achievable, since we are making an allowance for the case in which there is no known plan to achieve the remaining goals, but in a setting relying so heavily on user interaction, we cannot detect this with the limited information available mid-execution.

4.5 Our Approach as a form of Explanation-Based Learning (EBL)

There are numerous similarities between the goals and perspectives of the explanation-based learning approach and the approach we take in SPANDEX. In a sense, the methodology we propose in SPANDEX is a type of explanation-based learning. We can map SPANDEX into the general explanation-based learning perspective in the following manner:

1. **Goal concept.** In EBL, the goal concept is defined as a goal state, which is an incomplete world state specification. In SPANDEX the goal concept is also a goal state, but it has a slightly different role in the explanation process (see the section below which discusses the **explanation.**

2. **Domain theory.** In EBL, the domain theory consists of objects with properties, inference rules for inferring more about object relationships and properties, and problem-solving operators (activities). In SPANDEX the domain theory consists of the same types of entities. However, our inference rules are not represented explicitly as rules; rather such rules are embedded in actual object definitions in the object hierarchy. SPANDEX also has access to knowledge about agents, motivations, etc. Another difference is that the representation used by SPANDEX provides a richer constraint language to be used in schema definitions than the constraint vocabulary which seems to be available in the EBL systems previously reviewed.

3. **Single Training Example.** In EBL, the single training example is a sequence of operators (with some operators potentially missing) that represents the observed problem solving behavior of an agent. Similarly, in SPANDEX, the example is also an observed sequence of primitive actions which have been performed. Specifically, an example in SPANDEX is an ordered token string of action instantiations which thus far have been either executed by the planner or performed by the user, including the exceptional action as the last token. In the standard EBL approach, the observed sequence is given as a single input representing a complete plan. In contrast, SPANDEX is given only a partial sequence of primitive operators, as an indication of the incremental nature in which SPANDEX "observes" and attempts to "understand" the problem-solving behavior of an agent.

4. **Explanation.** In EBL, the explanation is an "operator sequence that solves a problem, together with an annotation which captures how the effects of one operator match the preconditions of another." An explanation includes expansion matching choices, and is meant to justify how the precondition of each operator is achieved,

and how the goal is achieved. Operators and states which don't "causally" support goal are eliminated. Thus, in EBL, an explanation is really a sort of *proof* of how the observed operator sequence solves the initial problem. In SPANDEX, at the point where an exception is encountered, we do not yet necessarily have in hand a complete sequence of primitive steps which define the final plan. Thus we cannot absolutely prove the the goal will actually be met. In SPANDEX, we attempt to use explanation to show that the observed partial action sequence (including the exception) *can be part of at least one possible plan network which is consistent with achieving the goal concept*. That is, as far as we can tell, this sequence of actions can fit into *a* plan network which can achieve the goal concept, though not necessarily the current plan network without alteration.

In SPANDEX, an explanation is contructed within the context of a current plan network, which is a complete plan specification with goal and activity nodes at varying levels of abstraction and predicted action nodes highlighted. A SPANDEX explanation is a new tree-like plan network which contains at the leaf level the token string of observed and performed actions (the initial training example). The inner nodes of the plan-network tree are nodes which are the more abstract specifications of the primitive plan steps, as well as unexpanded goal nodes. The root node of the explanation structure is the goal concept. This explanation structure contains causal links and is annotated with the justifications of the role of the unexpected action in the evolving plan. The previous state of the plan network may be referred to in the annotations, in order to express justifications such as which step in the previous plan network was replaced, what previous steps are no longer necessary due to the unexpected action, etc.

5. **Result.** The end result for both the EBL and SPANDEX approaches is a general schema of which the instance is just an example. The primary focus in the EBL work is on techniques for generalization, while our initial focus in SPANDEX is to develop a set of techniques for constructing different types of explanations. We do, however, recognize the need for generalization techniques in order to learn through the handling of exceptions, and perhaps will make use of existing methods such as those used by the EBL approach.

In summary, the goals we are trying to achieve in SPANDEX can be mapped fairly closely into an explanation-based learning perspective. However, we have identified the following as novel aspects to our problem and approach:

1. Our planning model is one of incremental planning and execution. Planning and execution are interleaved.

2. Our planning system is interactive, and this interface design imposes a cooperative framework upon the users and the system.

3. We assume an incomplete domain theory, and suggest the paradigm of negotiation to extend the theory. Negotiation supplements the more bounded task of *reorganizing* existing knowledge through automated explanation construction (shared in common with EBL).

4. We introduce agents as an important component of the planning model. Motivations for unusual agent behavior are identified. Negotiation must take place between agents regarding an "explanation."

5. We rely heavily on an abstraction/generalization hierarchy of activities (and objects) to construct the explanation of an exception. EBL does not seem to use this type of information explicitly during explanation construction, though similar reasoning seems to play a role during the generalization process.

In the next section, we present detailed examples of the detection and explanation of two of the possible exception types: an *action-not-in-plan* and an *unexpected-parameter* exception which causes a static object constraint violation. We give examples of structures used during explanation construction and show how the knowledge base may be examined by the exception analyst.

Chapter 5

Example

The taxonomy of exceptions developed in chapter 2 and the reasoning principles outlined in chapter 4 are further illuminated in this section through the provision of a detailed example. Our example domain is the world of real-estate; specifically the activities involved in buying a house. This domain was chosen not only because of its familiarity, but the activity of purchasing a home is one which is described naturally in terms of both procedures and goals, involves several active agents, and is likely to be fraught with exceptional occurrences. The procedure for buying a house is both simple enough for most people to understand and is complex enough to demonstrate the different types of exceptions that can arise and how they might be handled.

In the discussion of the following example scenarios, the reader should refer back to Figures 4.2 and 4.3 on page 20 in section 4.1 to regain familiarity with the overall task decomposition and domain entities.

5.1 Scenario 1: Action-Not-in-Plan

Suppose that the buyer has selected a house to buy, and the *purchase-and-sale-agreement* has been signed. The system next expects the buyer to go about obtaining a mortgage, an activity whose first step is to *go-to-bank*. Now, suppose that the execution monitor instead detects that the buyer's actual action was to *sell-stock*. Since the actual action is not among the expected-actions set, an exception has occurred. Further examination by the exception classifier reveals that the *sell-stock* action does not fit into any of the possible plan expansions of future steps, so it is not an *out-of-order* step. The *sell-stock* action is thus classified as an *action-not-in-plan* exception. An instance of an *exception record* (see Figure 5.1) is created to store information about the exception, and is attached as an annotation to the node representing the actual action *sell-stock1*.

The plan critic is then invoked to determine the effects of the exceptional action on the

LABEL:	exception-record1
EXCEPTIONAL-ACTION:	sell-stock
EXCEPTION-TYPE:	action-not-in-plan
PRECONDITION:	true
PROTECTED-STATES-VIOLATED:	nil
EXPLANATIONS:	explanation-record1

Figure 5.1: The exception record for Scenario 1

current procedural network. It is invoked with a specification of the effects of the actual action and performs the following two actions:

1. The precondition of the action performed by the user is checked. If it is not true, that information is entered into the *exception-record*. (In this case, the precondition of the *sell-stock* action, namely, that stock exists, is true.)

2. The effects of the actual action are examined in the context of the current procedural net to determine whether the assertion of the specified states will violate any protected states. If so, these violations are entered into the *exception-record*. (In this particular case, no protected states are violated.)

The exception analyst is invoked to search for explanations that would establish why a *sell-stock* action could be a valid action at this point in the plan. One heuristic guiding the search for explanations when an *action-not-in-plan* exception occurs is that the agent is attempting an alternative way of accomplishing the goal of either:

- An expected action, or
- An activity whose status is *in progress*[1].

The exception analyst forms a *candidate-list* by merging the expected-actions-list and a computed list of the in-progress activities. Each element of this list is compared to the actual action to determine substitutability, according to the tests specified in section 4.3.1.

In the current scenario, the *sell-stock* action is first compared to the *nearest*[2] action on the *candidate-list*, specifically the *go-to-bank* action. There is no significant taxonomic

[1]An activity is considered to be *in progress* if one or more of the steps in its expansion is on the *expected-actions-list*.

[2]The *nearest* action to an actual action is defined to be a primitive action which is among the *expected-actions-list*. The level of "nearness" decreases as we climb up the hierarchy representing the order of expansions which preceded the primitive level nodes.

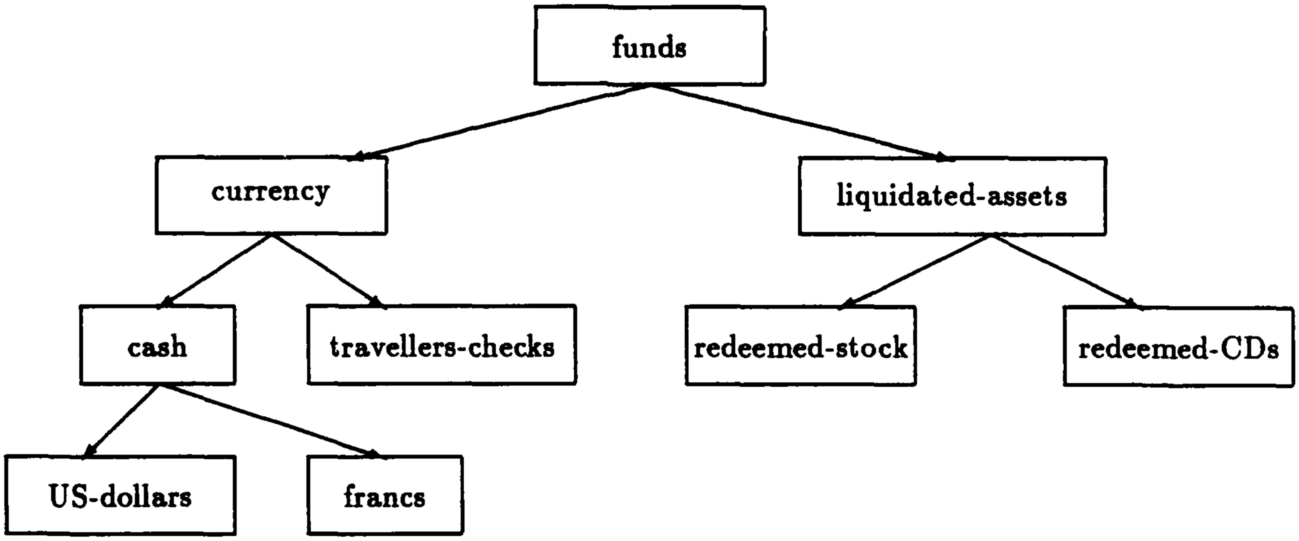


Figure 5.2: Portion of domain object hierarchy

relationship between the two actions, nor is there an overlap of the effects of the two actions. Comparison of the actual ***sell-stock*** action proceeds to the next nearest action on the candidate-list, the in-progress activity ***apply-for-mortgage***. Again, no significant relationships emerge. ***Sell-stock*** is next compared to ***get-mortgage***, which is another in-progress activity which subsumes apply-for-mortgage. There is no taxonomic relationship between these two activities, but the goals of the two activities are clearly related. Both activities goals establish the existence of some resource, namely ***funds*** in the case of ***get-mortgage***, and ***redeemed-stock*** in the case of the ***sell-stock*** action. Since the goal predicates are the same, it remains to be checked whether a significant relationship exists between the parameters of the two goal statements. The exception analyst notes that ***redeemed-stock*** is a specialization of ***liquidated-assets***, which in turn is a specialization of ***funds*** (see Figure 5.2) and thus "having redeemed-stock" is a sufficient operationalization of "having funds."

The above reasoning process represents the construction of one complete explanation of the role of an unexpected ***sell-stock*** at this point in the plan. Other possible explanations were examined and determined to be invalid[3]. All explanations investigated, both valid and invalid, are stored in the exception-record, within separate attached ***explanation records***. An explanation record contains the relationships and inferences that are central to the explanation. One completed record for the ***sell-stock*** scenario is shown in Figure 5.3.

In addition to recording the logical or heuristic basis for the semantics of the explana-

[3]The failed equivalency tests between the ***sell-stock*** action and ***go-to-bank*** and ***apply-for-mortgage*** actually represent potential explanations, although invalid given currently available information.

LABEL:	explanation-record1
EXCEPTION:	exception-record1
EXPLANATION-SUMMARY:	(TYPE: step-substitution; JUSTIFICATION: specialization (Type: goal-parameters; Values: redeemed-stock, funds))
PRIMARY-PLAN-MODIFICATION:	substitute(sell-stock1, get-mortgage1)
PLAN-MODIFICATION-SIDE-EFFECTS:	deactivate(go-to-bank1, apply-for-mortgage1, receive-mortgage-approval1)

Figure 5.3: Explanation-record for Scenario 1

tion, the exception analyst must also determine what changes may have to be made to the current procedural network structure in order to accomodate the exceptional action. In the case of the current scenario, since the equivalency of the *get-mortgage* action and the *sell-stock* action has been determined, the desired change is to substitute a node representing the *sell-stock* action for the node representing *get-mortgage*. However, since *get-mortgage* is a complex activity which is "in-progress," the expansion which is subsumed by this node must be eliminated from the procedural network. In other words, since the buyer has sold his stock as a method for obtaining funds, he is no longer expected to follow through with any of the steps involved in obtaining a mortgage. Therefore, the nodes which are subsumed in the hierarchical plan network wedge which is headed by *get-mortgage* (*go-to-bank, apply-for-mortgage, receive-mortgage-approval*) must be **deactivated**. This information is also stored in the explanation-record, and will be used by the routines which perform the actual modification of the procedural network, if explanation is later verified.

Once the set of explanations are constructed, the negotiator will engage the involved agents in a dialogue to either:

- Select one of the completed explanations, or
- Elicit more information from an agent to establish the validity of one of the partial explanations.

In this scenario, the buyer is the only agent involved in this exception, and chooses the explanation represented by explanation-record1 as a valid explanation. The underlying reason for the exception was an intended substitution, since the buyer intends to purchase the house with his own funds rather than assuming a mortgage. The procedural network modification routines are then invoked to make the changes to the current procedural network.

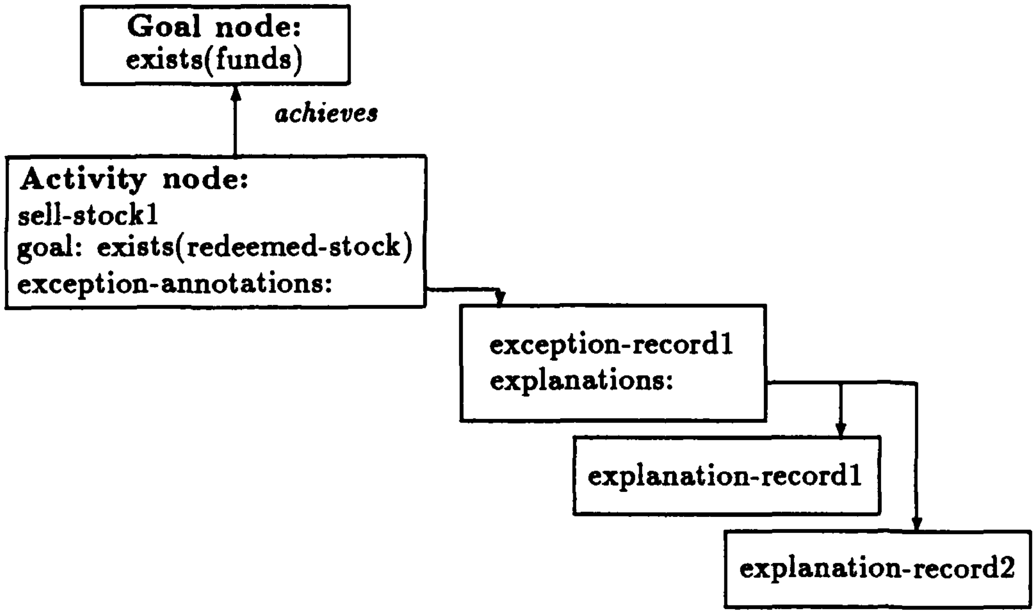


Figure 5.4: New procedural net, with attached explanation

The final explanation of the exceptional scenario consists of the new procedural network, incorporating the exceptional *sell-stock* action, which itself is annotated with *exception-record1* (containing pointers to explanation records including *explanation-record1*) (see Figure 5.4).

5.2 Scenario 2: Expected action, inconsistent parameter

Having illustrated the actions of the system upon encountering an *action-not-in-plan* exception, we now give an example of a case where the action type observed is as expected, but a parameter inconsistency is detected. In this scenario, suppose that the buyer is at the stage of filling out a *purchase-and-sale-agreement* and is executing the step *fill-out-form-field-1*, where the value of the field is constrained by the *form* object to be an *address*. Now suppose the action token observed by the execution monitor is of the correct type but the parameter provided by the buyer is actually "545-0609," which is a *phone-number*. An exception of type *unexpected-parameter* has occurred. This exception is further classified as one which causes a *static object constraint violation* since the *address* field of a *form* object has an attached constraint specifying that the value must be of type *address*, and

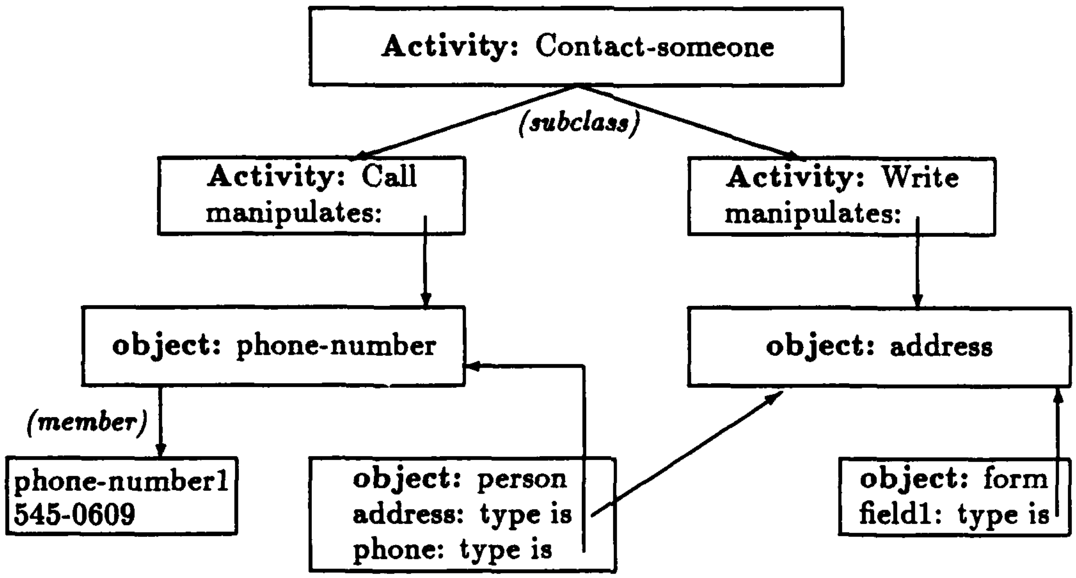


Figure 5.5: Fragment of knowledge base relevant to phone numbers and addresses

it is this constraint which has been violated.

Information about domain objects in the knowledge base is represented in a variety of ways. Objects such as *phone-numbers* and *addresses* are represented as structured entities, with well-defined fields and constraints. The activities which manipulate these objects can be found by following links, so the ways in which these objects are used can be easily determined. In addition, other links may specify the domain objects which these particular objects may be part of or related to (see Figure 5.5). The task of the exception analyst upon the detection of an *unexpected-parameter* exception is to explore the connections between the unexpected and expected parameter objects in an attempt to discover a significant relationship which would justify the parameter inconsistency.

An intuitive explanation for this exception might be that the action of the buyer was a careless "slip" – that he meant to supply the *address* but inserted the *phone-number* by mistake, because of its close association. One approach might be to ask the agent to verify the suspected error, and revise his action. This tactic might be tried first, to save more costly explanation construction and verification.

Assuming that the agent does not acknowledge an error, the first hypothesis to pursue in an attempt to explain an exception of this type is the following: perhaps the unexpected parameter value was meant as a replacement for the expected value. The exception analyst pursues a number of paths in an attempt to substantiate this. The simplest ex-

planations are investigated first. For example, perhaps a *phone-number* is a specialization of the *address* object, and is thus sufficient, as was deduced in Scenario 1. This is an invalid explanation, however, as can be seen by examining Figure 5.5. Other taxonomic relationships are investigated to establish partial explanations that might be completed and verified though additional processing and negotiation. For example, if an *address* could be shown to be a specialization of a *phone-number*, or if both objects were shown to be subclasses of a common object, it could be suggested that a more general object type (*e.g.* the *telephone* object or the common parent object, respectively) should have been provided in the *form* object's field constraint specification, rather than *address*. However, again, as can be noted by the reader, attempts to establish these relationships fail and thus the corresponding partial explanation structures are marked as invalid (become null explanations).

An alternative approach that would bypass interaction with the agent is for the exception analyst to investigate if the *address* can be deduced from the *phone-number* supplied, perhaps through a search of the *people* objects with the target *phone-number* and then retrieving the appropriate *address* from the *person* instantiation which produces a match. This search however, may be quite costly, and is thus only done in cases where the number of objects containing the field in question can be ascertained to be below a fixed limit. In this case, we may assume that the number of people instantiations in our knowledge base exceeds the limit, yielding a preference for attempting other remaining strategies first.

The actual usage of the objects in question is examined next, in hopes that the unexpected parameter object and that which was expected have a similar function within the plan. The following information can be gathered by the exception analyst:

> *Telephone-numbers* are used by the *call* activity, while *addresses* are used by the *write* activity. Both the *write* and *call* activities are specializations (level of nearness = 1, since they are at the same level of abstraction) of a more abstract *method-of-contact* activity. In other words, both of these objects are used by activities which are very similar in function.

Intuitively, we can assume that the buyer filling out the form was supposed to supply an address, so as to facilitate later attempts to contact him while the sale of the home is being negotiated. However, the buyer may be in the process of relocating and can only be reached by phone at his work number. The action of filling out the *purchase-and-sale-agreement* with a *phone-number* instead may have been an intentional action to supply the real-estate office with an alternative method of contact. This explanation is thus encoded into the exception-record and explanation-record shown in Figures 5.6 and 5.7, respectively.

When permitting the violation of a static object constraint, as is suggested by this explanation, an important consideration is that later steps of the plan may access the *address* field of the *form* involved and unexpectedly find a *phone-number*. For example,

LABEL:	exception-record2
EXCEPTIONAL-ACTION:	fill-out-form-field1 (type *phone-number*)
EXCEPTION-TYPE:	unexpected-parameter, static violation
PRECONDITION:	true
PROTECTED-STATES-VIOLATED:	nil
EXPLANATIONS:	explanation-record2

Figure 5.6: The exception record for Scenario 2

LABEL:	explanation-record
EXCEPTION:	exception-record
EXPLANATION-SUMMARY:	(TYPE: parameter-substitution; JUSTIFICATION: similar-function (Type: step-parameters; Values: phone-number, address))
PRIMARY-PLAN-MODIFICATION:	substitute(fill-out-form-field1(phone-number, address)
PLAN-MODIFICATION-SIDE-EFFECTS:	attach-usage-demon(form1, field1)

Figure 5.7: Explanation-record for Scenario 2

once the *purchase-and-sale-agreement* has been signed by the seller, the real estate agent may want to mail a copy to the buyer, and will be confused when "545-0609" is retrieved as the address for sending the copy. Such future accesses must be warned of the inconsistency, so that procedures which may applied to the unexpected value (in this case, *U.S.-mail*) can be modified accordingly (*e.g.* the buyer must be *call*ed to come pick up the copy, rather than having it sent out). This special treatment is exemplified by the side-effect listed in 5.7 specifying the attachment of a demon to the *form* object in question[4].

Again, as in the first scenario, once the potential explanations have been constructed, control is passed to the negotiator to determine which of the explanations should be chosen or explored further, and what changes to the procedural net or knowledge base are necessitated.

[4]Other techniques for accomodating exceptions to constraints in databases, such as those described by Borgida [5], may be investigated for our use when handling this type of exception.

Chapter 6

Research plan

6.1 Review of the problem and our goals

At this point in this proposal, it may be helpful to review the motivating forces for the SPANDEX architecture and approach which we have presented. The following are the important aspects of our problem:

- The real world often does not operate in a "typical" fashion. It is difficult the anticipate all the possible ways of performing a task goal, and the environment is often dynamic. A goal that is largely overlooked in current planning systems is to be able to handle unexpected contingencies as they arise in a planning framework[9]. Current planners are too "breakable" in the face of unexpected events.

- Intelligent agents involved in plan execution are often important parties in a planning framework, and are frequently overlooked. The behavior of agents should be analyzed critically, especially when it is *inconsistent with system expectations*. Agents who interact with the system are important sources of knowledge for refining an incompletely specified domain.

- Agents perform purposeful actions; their behavior is seldom random. Previous systems have not addressed the notion of attempting to understand "erroneous" agent actions within a planning framework. Replanning is a reactionary approach that is not sufficient for sophisticated explanation of unusual occurrences and the corresponding modification of plans.

- A domain model that is the initial input to a planner is often incomplete and incorrect. Intelligent agents have effectively unlimited knowledge bases, but the knowledge is dispersed and not explicitly accessible. One way to have the system perform as

if it has access to what the user knows is to monitor and analyze exceptional user behavior.

In response to the characteristics discussed above, we would like to provide a "flexible" intelligent assistant to help agents perform tasks. The system should be able to go beyond "reactionary" approaches when encountering unexpected events or states, and attempt to **understand** possible **intent** behind the exception and how it might be incorporated into a consistent plan. This work constitutes an attempt to bridge the gap between the knowledge guiding system behavior, and that guiding agents' behavior. The eventual goal is to have the system domain model better approximate the domain model of the intelligent agents.

Therefore, the goal of this research is to demonstrate that we can continue the planning and execution of a task in a consistent and uninterrupted fashion upon encountering an exception, with the help of the following:

- a careful look at the types of exceptions that can occur in an interactive planning framework,
- an understanding about what motivates users to perform actions which are different from system expectations,
- a rich integrated representation for domain activities and objects, and
- a set of algorithms for control exploration of that representation to find meaningful "explanations" for exceptional behavior.

Furthermore, we would like to show that this approach produces a planning system which is less brittle and more efficient than previous planners which adopt a simpler re-planning approach when encountering exceptional occurrences. Finally, we hope to demonstrate that this approach produces a system which "learns" about alternative ways to complete task goals, and is able to use this new knowledge during future planning activities.

6.2 Demonstration of thesis

In this section, we outline the aspects of this overall design which we expect to implement, and discuss ways of evaluating the approach.

6.2.1 Implementation

We will build a system called SPANDEX[1] which will demonstrate the approach we have presented for handling exceptions. At this point, we are not focusing on the implementation of the replanning or plan critic modules. We are assuming behavior in this regard which is typical of [54]. We are also not concentrating on a detailed design or implementation of the negotiator, since the negotiator alone is a substantial project. In the initial stages of the implementation, we will be concentrating on the exception classifier and the exception analyst. Later stages will deal with the implementation of the module which will incorporate changes into the existing knowledge base, as suggested by the output of the negotiation phase. The SPANDEX system will be implemented using KEE on a Texas Instruments Explorer.

Our planner

The SPANDEX system is part of a comprehensive planning system (POLYMER), the kernel of which is largely implemented [11,12]. POLYMER is a system designed to support the activities of decision-making, communication, and information manipulation that are inherent in a cooperative work environment. The object-management subsystem (OMS) contains detailed knowledge about the domain activities, the objects they create and manipulate, and the people or "agents" who are responsible for their execution. The OMS is object-oriented since entities have procedural attachments which are inherited through abstraction hierarchies. The planner is interactive, treating the user as an active agent in planning decisions, and user actions incrementally provide constraints to guide the planning process. The activity description language used by POLYMER is based on formalisms used by other planning systems [8,10,53]. The representation provides mechanisms to express causality and includes a general looping mechanism for expressing different forms of iteration. The details of the POLYMER planner and OMS can be found in [31].

The basic cycle of POLYMER for a single user is as follows:

1. A goal is posted by the application (upon encountering a user action).
2. The planner expands the goal into a partial ordering of activities and subgoals, constructing the set of actions which can occur next in the form of an *expected-actions-list*.
3. The execution monitor selects an action from the *expected-actions-list* and sends a message to an active-object (to the user if it is an action which must be performed by the user, otherwise to a tool object).

[1] System for Planning AND EXception handling

4. The execution monitor compares the actual action taken with the set of actions on the *expected-actions-list*, and if a match is not found, the exceptional handling mechanism is invoked.

5. Once an explanation has been negotiated successfully (or if a match was found from step 5 above), control is returned either to: a) Step 3 if more actions remain on the *expected-actions-list*, or b) Step 2 otherwise.

Exception classifier

We should mention that we are not concerned the issues involved in the actual monitoring of user actions or world state changes. SPANDEX assumes that an action or state change has been detected, and all system activity proceeds from the point where SPANDEX is invoked with a well-formed formula or an action token which is inconsistent with the planner's expectations.

In the implementation of the exception classifier, we expect some of the exception types to be classified in a straightforward fashion, specifically the unexpected-parameter, user-assertion, external exception, and repeated-step exceptions. However, distinguishing out-of-order exceptions with action-not-in-plan exceptions is less trivial. We will need an efficient strategy that will most likely involve an unintelligent expansion to produce an approximate computation, and we will have to provide for the possibility of inaccuracies.

Exception analyst

The implementation of the exception analyst will incorporate:

1. A set of heuristics for mapping from an exception class (as determined by the exception classifier) to probable intents on the part of the user,

2. A set of heuristics for mapping from (exception-class, intent) pairs, as determined in (1), to search paths to guide the determination of the relationship of the exception to the current plan. We will need precise methods for controlling the search for explanations. Paths which may result in complete explanations should be pursued first, but when there are many possible paths to pursue (for example, if there are several expected actions to consider, or if the network encompasses many levels of expansion which may need to be considered), we will need to establish cut-off points.

3. A set of algorithms which operationalize the search strategies suggested in (2). These search strategies correspond to a taxonomy of explanation types. The execution of the appropriate algorithms will result in the creation and completion of exception-records and explanation-records to record the results of the processing. Before the actual implementation begins, we need to better formalize the types of explanations,

based on the examples given in chapter 5. This will be done in tandem with a more complete design of the basic explanation-record and exception-record data structures.

Explanation Generalizer

After the initial prototype has been built which constructs explanations for exceptional occurrences, we will need a method for generalizing a verified explanation to provide the system with the capability to learn from experience. We are looking critically at existing algorithms from current work on explanation-based learning and generalization and will attempt to adapt them to handle SPANDEX explanations.

Knowledge base modifier

As already mentioned, we are bypassing the actual implementation of the negotiation module, and will concentrate on attempting to integrate the changes which result from negotiation into the existing knowledge base. These changes fall into two classes:

- An exception may be justified by the gathering of information which was already in the knowledge base. In this case, the information about how the plan was accomplished may be better represented for future use so that the deductive process need not be repeated. The output of the negotiator, in this case, is simply the appropriate agents' verification of information discovered by the exception analyst.

- When the exception analyst was unable to deduce a complete explanation of an exception, further information may be acquired from the relevant agents via the negotiator. This *new* information should be reflected in the knowledge base through addition or change.

In either case, we are assuming that the negotiator has provided us with the appropriate changes to be made to the knowledge base, along with an indication of whether they reflect newly acquired or inferred knowledge. There are several issues involved in making changes to complex knowledge bases and ensuring the proper propogation of changes, which we plan to investigate [4].

6.2.2 Experimentation

We plan to investigate system performance by monitoring its behavior on a set of scenarios from the real estate domain. We intend to conduct a series of interviews with a mortgage officer in a local mortgage institution. The initial interview will concentrate on extracting an initial set of typical plans for processing a mortgage application, embodying the set of domain action primitives. Successive interview(s) will be aimed towards gathering a set of

realistic scenarios that demonstrate exceptions and interruptions that can occur during the mortgage process, including exceptional behaviors on the part of the mortgage institution, assessor, prospective buyer, seller, etc. This discussion of exceptional behavior will also be used as a metric regarding the completeness of our taxonomy of exception types.

6.2.3 Goals

The following are the goals which we would like to achieve in this implementation:

- The exception classifer should always be able to classify an action which doesn't match one of the expected actions into an exception type in our taxonomy.
- Our approach should be demonstrated to succeed when simple replanning would fail.
- The heuristics used by the exception analyst to select among its strategies for resolving exceptions should demonstrate an improved efficiency over a blind "try-everything" method. In other words, an explanation for the exception should be discovered with less search when using the heuristics than when a more random approach is employed.
- The exception analyst should be able to apply knowledge which is dispersed throughout the knowledge base to explain an exceptional scenario which defies the more explicit activity description.

6.2.4 Beyond the initial prototype

In addition to the initial prototype which will implement the exception classifier and the exception analyst, the remaining time and energy will be devoted in the following directions:

- We need to develop strategies for knowing when and how to generalize an explanation. As part of this, it must be determined when static knowledge should be changed (as opposed to the dynamic plan network instantiations) and which features should be abstracted from the explanation for generalization. We are examining other approaches for adaptation to this work.
- Any changes which result from the negotiation process should be made to existing domain knowledge in such a way that the knowledge base remains consistent.
- The approach outlined thus far generally assumes that inconsistent behavior on the part of an agent is generally purposeful. However, we also know that humans are prone to making errors, and a more extensive approach would consider models of errorful behavior as well.

- The negotiator module is a substantial part of the system which will remain to be designed in detail and implemented.
- We are initially considering the analysis of an exception in the context of a single overall task. It is much more complex to consider the analysis of the exceptional behavior in the context of multiple ongoing tasks which are being multiplexed among the agents, and this would be an issue to address in the future.

6.2.5 Implementation schedule

This final section contains the schedule we have established for the implementation of this research. The elements of functionality which are designated by the numbers in Figure 6.1 are the following:

1. Data structure definition
   (a) Explanation structures
   (b) Exception records
   (c) Explanation type hierarchy
2. Exception classifier (simple version, without search)
3. Exception analyst algorithms – Phase 1 (a subset)
4. Exception classifier (with search strategies)
5. Exception analyst algorithms – Phase 2 (more complex algorithms included)
6. Facility to handle the completion of partial and null explanations
7. Interface
   (a) Show explanations
   (b) User choice
   (c) Completion of explanations
8. Generalization
9. Knowledge base modifier
10. Experimentation

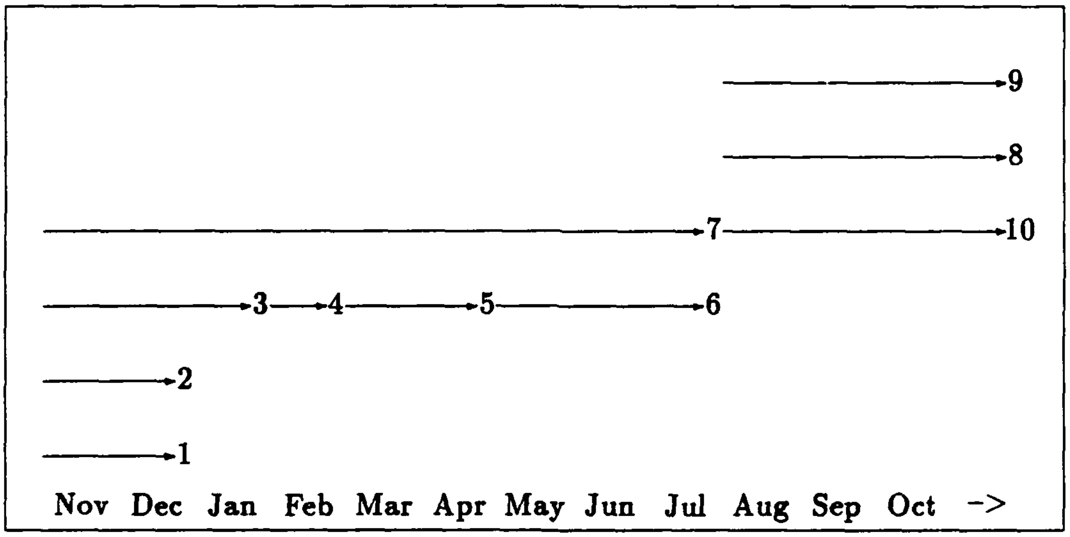


Figure 6.1: Implementation schedule for SPANDEX system

Bibliography

[1] Alterman, R. "An Adaptive Planner", *Proceedings of AAAI-86*, 65-69, 1986.

[2] Barber, G. "Supporting Organizational Problem-Solving with a Workstation," *ACM Transactions on Office Information Systems*, 21, 1 (1983), pp. 45-67.

[3] Alterman, R. "Issues in Adaptive Planning," Tech. Report No. UCB/CSD 87/304, Computer Science Division, University of California, Berkeley, California, July 1986.

[4] Banerjee, J., Chou, H., Garza, J., Kim, W., Woelk, D., Ballou, N. "Data Model Issues for Object-Oriented Applications," *ACM Transactions on Office Information Systems,* Vol 5, No. 1, January 1987, pp. 3-26.

[5] Borgida, A., and K.E. Williamson, " Accommodating Exceptions in Databases, and Refining the Schema by Learning from Them," *Proceedings of the Very Large Data Base Conference*, 1985, pp. 72-81.

[6] Broverman, C,; Croft, W.B. "A Knowledge-based Approach to Data Management for Intelligent User Interfaces", *Proceedings of VLDB 11*, Stockholm, 96-104, 1985.

[7] Broverman, C.A., Croft, W.B. "Exception Handling During Plan Execution Monitoring," *Proceedings of the Sixth National Conference on Artificial Intelligence*, July 1986, Seattle, WA.

[8] Broverman, C.A., Huff, K.E., Lesser, V.R. "The Role of Plan Recognition in Design of an Intelligent User Interface", *Proceedings of IEEE Conference on Systems, Man, and Cybernetics*, 863-868, 1986.

[9] Chapman, D. "Planning for Conjunctive Goals," *Artificial Intelligence*, 32, 1987, 333-377.

[10] Croft, W.B.; Lefkowitz, L.S. "Task Support in an Office System", *ACM Transactions on Office Information Systems*, 2: 197-212; 1984.

[11] Croft, W.B., Lefkowitz, L.S. "Knowledge-Based Support of Cooperative Activities," *Proceedings of the Hawaii International Conference on System Sciences*, to appear.

[12] Croft, W.B. "Representing Office Work with Goals and Constraints," *Proceedings of the IFIP Workshop on Office Knowledge*, Toronot, 1987.

[13] DeJong, G.F. "Generalizations Based on Explanations," *Proceedings of the Seventh Internation Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August 1981, pp. 67-70.

[14] DeJong, G., Mooney, R. "Explanation-based Learning: An Alternative View," *Machine Learning*, 1, 1986.

[15] DeJong, G. "An Approach to Learning From Observation," Machine Learning, Vol. II. Michalski, Carbonell, Mitchell, eds., Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986, pp. 571-590.

[16] Dietterich, T.G., "Learning at the Knowledge Level," *Machine Learning*, 1:287-315, 1986.

[17] Doyle, R.J. "Constructing and Refining Causal Explanations from an Inconsistent Domain Theory," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp.538-544.

[18] Doyle, R.J., Atkinson, D.J., Doshi, R.S. "Generating Perceptoin Requests and Expectaions to Verify the Execution of Plans," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp.81-87.

[19] Fikes, R.E. "A Commitment-based Framework for Describing Informal Cooperative Work", *Cognitive Science*, 6: 331-347; 1982.

[20] Fikes, R. Hart, P. and Nilsson, N. "Leaning and Executing Generalized Robot Plans," Artificial Intelligence, 3:2, pp. 251-258, 1972.

[21] Hammond, K.J. "Learning to Anticipate and Avoid Planning Problems through the Explanation of Failure," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, zAugust 1986, pp.556-560.

[22] Hammond, K.J. "Explaining and Repairing Plans that Fail," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August 1987, Milan, Italy, pp.109-115.

[23] Hammond, K.J. "CHEF: A Model of Case-based Planning," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp.267-271.

[24] Hammond, K.J. "The Use of Remindings in Planning," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, 1986, pp.442-451.

[25] Hammond, K.J. *Case-based Planning: an Integrated Theory of Planning, Learning and Memory.* Ph.D. Thesis, Yale University, 1986.

[26] Hayes, P.J. "A Representation for Robot Plans", *Proceedings IJCAI-75*, 181-188, 1975.

[27] Hollnagel, E. "Action Not as Planned: The Phenotype and Genotype of Erroneous Actions," draft, Computer Resources International, Copenhagen, Denmark, 1987.

[28] Kedar-Cabelli, S. "Purpose-directed Analogy," *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, August 1985, pp. 150-158.

[29] Kolodner, J. L., Simpson, R.L. and Sycara-Cyranski, K. "A process Model of Case-Based Reasoning in Problem Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985.

[30] Lebowitz, M. "Complex Learning Environments: Hierarchies and the Use of Explanation," *Proceedings of the 1985 International Machine Learning Workshop*, Skytop, PA, June 1985, pp. 110-112.

[31] Lefkowitz, L.S. and Croft, W.B. " An Interim Report on the POLYMER Planning System," COINS Technical Report, University of Massachusetts, Amherst, Ma. 1987.

[32] McDermott, D.V. "Planning and Acting", *Cognitive Science*, 2, 1978.

[33] Minton, S. "Selectively Generalizing Plans for Problem-Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 596-599.

[34] Mitchell, T.M., Keller, R.M., Kedar-Cabelli, S.T. "Explanation-based Generalization: A Unifying View," *Machine Learning*, 1:47-80, 1986.

[35] Mitchell, T., Mahadevan, S., and Steinberg, L.I. " LEAP: A Learning Apprentice for VLSI Design," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 573-580.

[36] Mooney, R. and DeJong, G.F. "Learning Schemata for Natural Language Processing," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985.

[37] Mooney, R., Bennett, S. "A Domain Independent Explanation-Based Generalizer," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555.

[38] Nilsson, N.J. *Principles of Artificial Intelligence*, Tioga, Palo Alto, California, 1980.

[39] O'Rorke, P. "Generalization for Explanation-Based Schema Acquisition," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 260-263.

[40] Pazzani, M.J. "Explanation and Generalization Based Memory," *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, August 1985, pp.323-328.

[41] Pazzani, M., Dyer, M., Flowers, M. "The Role of Prior Causal Theories in Generalization," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 545-550.

[42] Rajamoney, S., DeJong, G., Faltings, B. "Towards a Model of Conceptual Knowledge Acquisition Through Directed Experimentation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 1985, pp. 688-690.

[43] Rasmussen, J. "What Can Be Learned from Human Error Reports?" In K. Duncan, M. Gruneberg, and D. Wallis (Eds.), Changes in Working Life. John Wiley: London. 1980.

[44] Reason, J. "Actions not as Planned: The Price of Automatization," In G. Underwood and R. Stevens (Eds.), Aspects of Consciousness, Vol. 1. Psychological Issues. Academic Press: London. 1979.

[45] Rissland, E. L., and Ashley, K. *HYPO: A Case-Based Reasoning System*. 1987.

[46] Sacerdoti, E.D. *A Structure for Plans and Behavior*, Elsevier North-Holland, Inc., New York, NY, 1977.

[47] Schmidt, C.F., and Sridharan, N.S. "Plan Recognition Using a Hypothesize and Revise Paradigm," *Proceedings of IJCAI-77*, Cambridge, Ma. pp.480-486.

[48] Silver, B. "Precondition Analysis: Learning Control Information," Machine Learning, Vol. II. Michalski, Carbonell, Mitchell, eds., Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1986, pp. 647-671.

[49] Simpson, R. L. *A Computer Model of Case-Based Reasoning in Problem Solving*. Ph.D. thesis, Report No. GIT-ICS-85-18, School of ICS, Georgia Institute of Technology, Atlanta, GA., 1985.

[50] Sussman, G.J. A Computer Model of Skill Acquisition, Elsevier-North-Holland, New York, 1975.

[51] Tate, A. "Generating Project Networks", *Proceedings IJCAI-77*, Boston, 888-893, 1977.

[52] Tenenberg, J. "Planning with Abstraction", *Proceedings of AAAI-86*, 76-80, 1986.

[53] Wilkins, D.E. "Domain-independent Planning: Representation and Plan Generation", *Artificial Intelligence*, 22: 269-301; 1984.

[54] Wilkins, D.E. "Recovering from Execution Errors in SIPE", SRI International Technical Report 346, 1985.

[55] Wilkins, D., Clancey, W. and Buchanan, B. "ODYSSEUS: A Learning Apprentice," *Proceedings of the 1985 International Machine Learning Workshop*, Skytop, PA, June 1985, pp. 221-223.

Appendix 5-I

TOWARD A GENERAL ARCHITECTURE FOR INTELLIGENT TUTORING SYSTEMS

Tom Murray [1] [2]

COINS Technical Report 87-89

[1]This work was done in conjunction with the Cognitive Studies of Computers in Learning Program, at the UMass School of Education, and with the Intelligent Tutoring Systems Group, in the UMass Department of Computer and Information Science.

[2]This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008.

Abstract

This paper discusses design issues for intelligent tutoring systems, and suggests a general and extendable architecture for program control and knowledge representation. The main focus is on how discourse, teaching, and diagnostic rules, along with the information needed to implement these rules, can be acquired, represented, and coordinated. First, the state of the art in ITS design, important problems, and research goals are discussed. Then, a rule-based, goal-driven architecture is suggested which facilitates an evolving knowledge base, intelligent planning of tutoring actions, dynamicly shifting between different tutoring styles, and the maintenance of student and discourse models. Finally, the principles underlying knowledge acquisition and the identification of design constraints are discussed, and a step-by-step methodology is given. Selected sections of the paper serve well as an introduction to, or review of, the important design issues in ITS.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 ITS systems state of the art

It has been proposed and verified in limited instances that computers have the potential to act as powerful intelligent tutors–to have an expert level understanding of the the knowledge in their domain, an understanding of the pedagogical aspects of that domain, and an ability to tailor instructional actions, on the fly, uniquely for different students and different instructional contexts (Wenger 85, Clancey 86, Sleeman & Brown 85). It is also becoming increasingly clear that computer tutors can provide instructional environments, such as graphic simulations and large structured data bases, that are not obtainable without the use of computers (Burton 86 and diSessa 86). Research in ICAI (Intelligent Computer Aided Instruction) has made significant progress over the last 10 years, but thus far very few ITS's (Intelligent Tutoring Systems, synonymous here with ICAI) have been built which have the scope and the robustness to have been tested extensively in actual academic settings. This is not surprising, considering the complexity of the tutoring problem, which requires state of the art AI methodologies in knowledge representation and acquisition, control, communication, interfaces, and machine learning. Each attempt to use a new ITS system on students uncovers new problems or questions for the pedagogical and cognitive theories in the domain used. All this not withstanding, several sectors of our society, strapped with severe educational and training problems, anxiously await the maturation of the field, and more concrete evidence that it can soon deliver the promise of its touted potential.

1.2 A common ground is needed

The issues facing the builders, designers, and evaluators of ITS systems are many and complex. As the number of ITS systems being proposed and built increases, so does the need for a common ground on which to exchange ideas and results. It is much too early to expect substantial commonality amongst the philosophical and strategic approaches in such theoretically alive areas as pedagogy, knowledge representation, and student modeling, but it is not premature to move toward some common terminological frameworks and guidelines with which to exchange and critique the various theories, tactics, and computer systems in ITS research.

1.3 Purpose of the architecture

This paper has two purposes: first, to present a general control and knowledge representation architecture for ITS systems. And second, to provide methodological guidelines for designing ITSs and analyzing ITS needs in any domain within the framework of the architecture.

The architecture proposed in Chapters 4 through 6 is in part an attempt at generalizing and synthesizing the implicit and explicit components of existing tutoring systems. Some of its features suggest reatively new ways to use existing AI technologies in ITS design. The architecture can be viewed as a "shell" within which to encode the specific information for arbitrary tutoring systems. As such, it is a step toward an authoring system tool for building ITSs.

This is a report on work in progress, and some of the ideas herein are under-constrained. The architecture has not yet been fully implemented, and the details presented in this paper are sure to evolve as we implement it. It is hoped that the issues raised and the solutions proposed will be a useful starting point for further research, even if some of the details fall be the wayside. As an attempt at looking at the "big picture" of ITS design, it focuses on how knowledge is used to control the actions of the system, and on how to modularize the various components within a framework that encourages well structured communication between them. As such, parts of this paper may also serve as an introduction to the important issues in the field, and an overview of the state of the art in ITS system design.

1.4 Purpose for the guidelines

Even if a generalized ITS shell or authoring system (as suggested above) existed, it would be difficult to utilize it effectively. The difficulties of specifying,with computational precision, domain knowledge and tutoring knowledge for such a system are prohibitive for all but the most experienced research teams (given the current state of the art). Two things are needed to ameliorate this problem: an authoring or knowledge acquisition interface which makes it feasible to have pedagogical and domain experts (as opposed to computer scientists) take a larger part in the design of the knowledge base, and methodological guidelines or accepted lore about the analysis of an instructional domain to produce an ITS specification. Chapters 7 and 8 propose such methodological guidelines. (We do not address the issue of knowldge acquisition interfaces directly in this paper.)

1.5 Overview of the paper

The intended audience for this paper are those involved in ITS research. We assume the reader has had some introduction to the field of intelligent tutoring systems, the more popular ITS systems described in the literature (see Sleeman & Brown 82 and Wenger 86), and an acquaintance with basic artificial intelligence concepts such as frames, production rules, inheritance, etc. (see Barr, Cohen, & Feigenbaum 82, Winston 84B). It is not intended to extol the virtues and potentials of Intelligent Tutoring Systems, though they are numerous and exciting. We assume the reader is familiar with the potential and proven capabilities and refer her to (Sleeman & Brown 82, Wenger 86, or Yazdani 86). Following is an overview of the chapters with suggestions for the reader.

Chapter 2
: A discussion of the issues and problems in the field which motivate the rest of the paper. If you agree with the tenant that a general architecture is needed, this chapter could be skimmed.

Chapter 3
: Gives a top level "systems view" of tutoring; an analysis of the human act of tutoring or teaching in terms of its functional components and the kinds of information exchanged between those components. The control and knowledge representation issues introduced here are

expanded in later chapters. The chapter should be light reading by those familiar with various ITS systems. For those less familiar with the field it may provide a good introducition.

Chapter 4

Knowledge representation issues. A description of the contents of the component data bases, and some design considerations; a bit more technical than the previous chapters. A passing familiarity with AI knowledge representation issues is assumed.

Chapter 5

An analysis of top level control issues; goal specifications of a tutoring session, including such elements as teaching goals, curriculum, and learning environments. No AI knowledge is needed.

Chapter 6

An analysis of the dynamic, low level control issues; components for dynamic decision making aspects of tutoring systems. We proposes a general control architecture for coordinating the types of information discussed in chapters 4 and 5. This is the most technical chapter. A passing acquaintance with AI control issues is helpful, but some of the issues are explained in detail. (The sections on disadvantages of vanilla rule-based control architectures may be skipped by those familiar with these issues.)

Chapter 7

A suggested methodology for tutoring system design. Chapters 4 through 6 were concerned with the knowledge representation and control structures containing the tutoring expertise. This chapter shows how the requirements of the various components of the system constrain the design process. A suggested scheduling of design activities is given. It is not a technical discussion, but assumes familiarity with the components of the architecture introduced in previous chapters.

Chapter 8

A knowledge engineering analysis of the factors, assumptions, and principles behind tutoring system design decisions, giving examples from existing systems. A methodology is suggested for analyzing the requirements of an arbitrary domain to generate overall goals and constraints for an ITS.

Chapter 9

Conclusion. Summary of the major points, and a summary of which aspects of the paper are considered contributions. Finally, some thoughts on the future (of ITSs).

Chapter 2

BACKGROUND

We will not attempt to provide the reader with a summary of ITS research projects and issues (see Sleeman & Brown 82 and Wenger 85). In this chapter we will first list some of the problems facing researchers in the field, which the paper addresses. Next we will list research goals motivated by these problems. Finally we will outline the solutions and suggestions addressed in this paper.

2.1 Problems and issues

2.1.1 ITS systems often have little pedagogical foundation

Many ICAI systems (synonymous with ITS's) have been built which incorporate little understanding of how successful humans tutors teach. Such programs are designed from the intuitions of a small number of AI researchers, often without collaboration with an expert teacher working in the field, and without extended testing of the tutoring strategies or assumptions implicit in the system on real students early in the design phase. Brown et. al. (page 279 of Sleem›n & Brown 85): "...the work that went into SOPHIE II and SOPHIE III c explanation put the cart before the horse. We had no adequate theory of what it meant to understand a circuit and hence no well defined "target" model of what we wanted the student to learn." Clancey, in (Clancey 86 page 43) says "The student's knowledge (was) very different from MYCIN." Then he goes on to explain new insights in the organization and use of expert knowledge in the MYCIN-GUIDON- NEOMYCIN-HERACLES series of AI programs. But he says little about the process of learning the expert's knowledge. So far none of the tutors in the GUIDON

series have been shown to work with students, suggesting the same problems articulated by Brown above.

2.1.2 Each new system is built from scratch

Many intelligent tutors are being built and proposed, most of them designed from scratch. It is rare to see the control or knowledge representation schemes from one research group being used by another. General guidelines for ITS system design do exist, such as identifying the major components (student model, domain model, etc.— see Yazdani 86), but there are few specific guidelines on how to implement these components. Systems for organizing tutoring rules have been proposed (Clancey in Sleeman & Brown 85, Woolf 84A, Anderson, Boyle, & Reiser 85), but these systems have not been adopted. This may be due to the fact that they are not presented in the context of a generalizable system, or due to the tendency of AI researchers to "roll their own" rather than incorporate existing methodologies.

2.1.3 Existing ITS systems have limited scope

The majority of ITS systems have been designed to teach in limited domains and within limited pedagogical styles. For example, SOPHIE could only teach using a couple of electrical circuits, the BUGGY (Burton 82) and SIERRA (Van Lehn 83) projects deal exclusively with the subtraction skill. Focus on learning goals at the curricular level is rare (but see Anderson & Reiser 85, and Shute & Bonar 86). In most cases this is necessary because the research efforts have focussed on difficult theoretical issues, such as student modeling or qualitative simulation. But there is need for research on how ITS systems can be used to teach (or help teach) entire curricula using a variety of instructional environments and teaching styles.

2.1.4 ITS's are hard to evaluate

Current ITS systems are hard to evaluate comparatively (i.e. one to another) (see Soloway & Littman 86). Very few have left the lab to be used on large numbers of students where effectiveness can be evaluated statistically. The pedagogical and psychological assumptions behind the rules used in tutoring systems are largely implicit, and differ with each system (see Anderson, Boyle, Farrell, & Reiser 84, Collins 77, and Larkin 83 for examples of rules systems). This makes it hard to compare different systems, and hard to evaluate why a system failed to teach (Was it due to shallowness of the

domain knowledge representation, inadequacy of tutoring rules, or implicit theoretical assumptions?)

2.2 ITS research goals

2.2.1 More pedagogical and psychological foundations

Pedagogical considerations should be the motivating factors for designing ITS systems for specific domains. A theory of learning in the domain, and priorities concerning what types of learning and knowledge are most important for performance in the domain, should proceed the design of the tutor. ITS system designers need to work closely with experts in teaching during the design and evaluation phases of building an ITS. Ideally, the pedagogical assumptions should have been verified to some extent before the effort of building a tutoring system is expended (although it does appear to be very difficult to get conclusive statistical results on any sufficiently specific theory of learning or teaching). Also, assumptions made about learning and teaching in the domain should be made explicit, for the purposes of assigning credit and blame after evaluative experiments, and so that the system can be compared with other ITS systems. Each important design decision and each high level tutoring rule should be annotated with the pedagogical or psychological assumptions it embodies.

2.2.2 Flexible reusable ITS shells

The computer can provide unparalleled learning environments (Papert 80, Burton 86). Instructional strategies found effective in human tutoring can not be assumed to work in computer tutoring (although they must serve as a starting point). It is not possible to fully evaluate the effects of these new environments with off line testing. New environments or tutoring rules will need to be reasearched to gain confidence concerning a system's teaching potential. The goal of designing ITSs using sound pedagoly is hampered by this lack of information about how people learn in these new environments. ITS systems should be powerful enough to embody a variety of tutoring strategies, and allow easy reconfiguration of these strategies in research settings. Therefore, systems should be flexible enough to easily add or modify domain knowledge, or even plug in a pedagogically similar domain. Systems with this kind of generality facilitate research on computer aided learning, ameliorate the problem of starting from scratch every time, and steepen the

research learning curve.

2.2.3 Toward ITS authoring systems

It is not unreasonable, given the current state of the art, to design ITS systems which can be configured, modified, and/or tweaked by domain or pedagogical experts with training in AI concepts, but are essentially non-programmers. We are a far cry from designing anything like an ITS authoring shell, but there is much we can do in the way of making the workings of systems as transparent and modular as possible, in an effort to allow instructional experts to have a larger hand in ITS system design and evaluation. Perhaps we should build systems as if they were to be used or maintained by parties who are initially unfamiliar with the design decisions.

2.2.4 A more precise technical vocabulary

If, as is suggested above, designers of ITS systems routinely tried to consult instructional experts, and performed case studies of learning and teaching situations to verify the underlying assumptions of the tutoring rules used, it would be a difficult and ill-defined task because we do not have a concise vocabulary or ontology with which to discuss the principles involved in designing tutors. Anderson (in Anderson 84) says: "Instruction in general, and computer- based instruction in particular, is pretty much a black art."

If intelligent tutoring systems are to successfully emulate good human tutoring, we need to computationalize the rules of effective instruction which tutors know implicitly. A terminology with the required precision and completeness does not yet exist, but some are striving for one (Clancey 86B). Clancey (in Clancey 85, page 309) stresses the importance of terminology in the development process: "The very process of formalizing terms and relations changes what we know, and itself brings about concept formation." On the same page, he warns of "the difficulty of defining terms," echoing the growing concerns of all AI knowledge representation research.

2.2.5 Tutoring entire topics, breadth and depth

There is a need for more ITS research on systems capable of a more curricular focus, i.e. more focus on how to teach entire topic units or textbook chapters. For example, traditional science courses include lectures, labs, homework, discussion sessions, and exams, all within a (supposedly) well thought out sequence to topics to be covered. Even though there may be much to be

desired in the way science is traditionally taught, it is obvious that a variety of environments and tasks are needed for significant learning in any field.

Students should be presented with several instructional styles and/or learning environments in an attempt to teach concepts from several view points. Most existing systems, since they attempt to teach a very limited subject matter, use a limited number of tutoring rules and environments.

2.3 Solutions and recommendations addressed in this paper

ITS research is not plagued with numerous fundamental problems, as the above summary might suggest. In fact the field is making steady and exciting progress along many dimensions (for example, modeling qualitative reasoning, student plan recognition, and analysis of problem solving skill acquisition). The problems listed above are given only to motivate the subject of this paper: general architectures and guidelines for ITS design. Below we preview the research goal solutions which are addressed in this paper:

2.3.1 A general ITS architecture is proposed

A general architecture is proposed that addresses the needs of managing diverse types of domain-specific teaching knowledge and large sets of general tutoring rules. The system is modular and flexible, and should help with several issues. Modularity and re- usability will lessen the need to design systems from scratch. Clearly distinguished functional components will increase the effectiveness of evaluations and comparisons of systems, and clarify issues in the design phase. Also, flexible rule systems will allow experimentation with different configurations of tutoring rules and knowledge representation. A clear organization of the types of information used to drive the system should also make it easier for pedagogical experts who are not computer scientists to take part in the design of tutors.

2.3.2 An analysis of the ITS design process

Suggestions are given for an organization of design activities. The organization stresses the incorporation of sample tutorial dialogues (Chapter 7) and a domain analysis making explicit the educational, pedagogical, and psychological assumptions and goals of the research team (Chapter 8) which for the basis for global and local design decisions.

2.3.3 Some suggestions for terminological refinement

Interspersed throughout the paper are suggestions for more precise terminology and ontology dealing with knowledge representation, control, and design. Perhaps more important than the specific ontologies proposed, is the idea that one is needed, this paper giving an example case.

Chapter 3

ITS—A SYSTEMS PERSPECTIVE

3.1 The constructivist paradigm for learning

The "learner as container, teacher as filler" paradigm for instruction is giving way to a Piagetian inspired constructivist view (Piaget 72, Lawson 75, Confrey 85, Von Glasersfeld 78). The constructivist view is that one learns by constructing new knowledge from existing knowledge through the self-regulatory processes of assimilation and accommodation. In assimilation experiential information is incorporated into existing "schema" (patterns or groupings of information). Accommodation is the modification of schema (or instantiation of new schema) as a result of experiential information which does not agree with existing schema. Learning is not a passive process of imprinting incoming information in the brain, it is an active process of trying to maintain equilibrium in a changing environment. If we believe, as most educational psychologists and epistemologists do today, that learning is an active process, then teaching can be seen as assisting a student in the learning process. This contrasts with the more didactic belief that knowledge is "conveyed" or "given" to students, who, if they pay attention, passively absorb it. In terms of the constructivist paradigm, teaching consists of: 1. providing a motivating environment in which to learn, and/or 2. specific guidance, helping the student "travel" from one knowledge state to a more desired one. There is a range of strategies which span the spectrum from passive learning environments to actively guiding the student, and we will outline these in Chapter 5.

3.2 Teaching vs. tutoring

For most of this paper we will be using the terms teaching and tutoring interchangeably, but some discussion of these terms is warranted. Teaching, often connoting an academic context, usually implies a structured environment, where the teacher has a predetermined sequence of instructional goals and activities typically for the student to participate in or observe. The activities are for a class of students and there is little opportunity to tailor the activities or the teacher's discourse to individual needs. Tutoring connotes a one-on-one situation with less structure and much more flexibility for attending to individual needs. Often tutoring actions are motivated by what the student says she is having trouble with. Intelligent tutoring systems try to combine both teaching and tutoring. They can use a structured lesson plan, which has high level goals determining an overall sequence of topics to be covered and the activities the student will engage in; they can make fine adjustments or refinements to these high level lesson plans according to individual needs; and they can introduce activities or dialogues on the fly to respond to the requests and instructional needs of individual students. Hence forth in this paper, we will use both "tutoring" and "teaching" to mean a combination of all of the above capabilities in computer or human instruction.

3.3 Rules for tutoring

Wenger (86) equates tutoring with "knowledge communication", and defines it as "the ability to cause and/or support the acquisition of one's knowledge by someone else via the channel of a restricted set of communication operations". ITS systems try to model the knowledge communication ability of successful human tutors. If we are to build ITS systems that emulate human performance, we must assume that human tutors behave according to some implicit rules concerning what actions to take (or not take) in each situation. In figure 1.1 we diagram how information describing the current tutoring situation is used by the tutoring rules and updated by tutoring actions. This schematic is not intended to have deep psychological validity, but to suggest a way of organizing the information flow and control mechanism of a computer tutor so as to account for behavior we observe in human tutoring. The parts shown in this schematic can be found explicitly or implicitly in all existing tutoring systems, and much of the terminology for the components is fairly well accepted in the field.

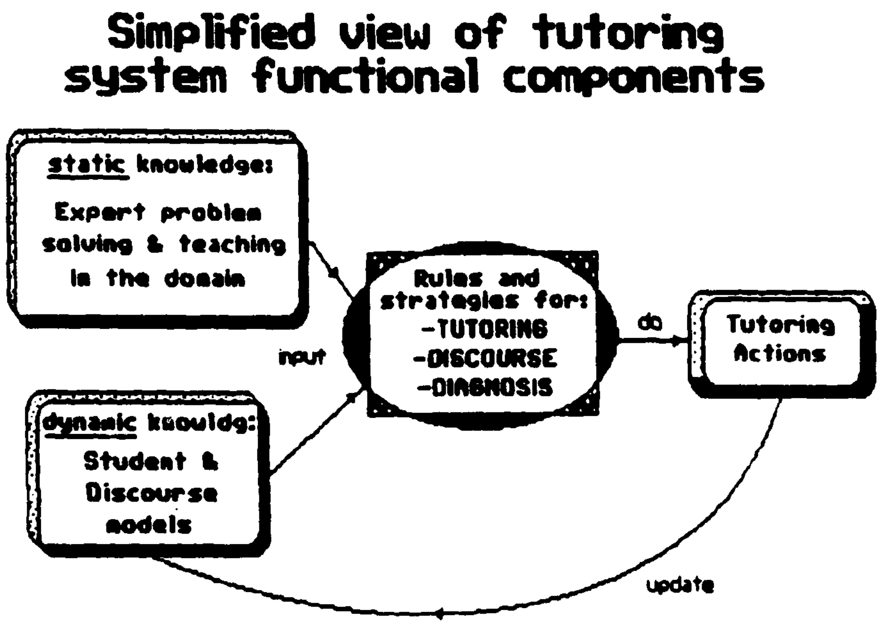


Figure 3.1: Simplified ITS functional components

The tutoring rules are typically of the form "IF <Situation> THEN <Action>", where the Situation specifies properties of the current tutoring context, and the Action specifies the appropriate action to take for that situation. (Such rules are also called production rules or productions.) Conceptually, these tutoring rules can be thought of as a mapping from all possible tutoring situations to all tutoring actions. The static information shown in figure 1.1 represents the tutor's permanent knowledge about problem solving and teaching in the domain being taught. The dynamic information represents constantly updated knowledge about the current situation, such as the tutor's model of what the student knows, the structure of the current discourse, and knowledge of the tutor's current instructional goals. The Situation (also called the condition or antecedent of the rule) is typically a complex combination of predicates, such as "if (the topic is just being introduced and (the student did well on the last topic or the student has 90 percent of the prerequisite concepts for this topic)) then ..." For now we will assume that the Action (also called the consequent of the rule) is the name of a single action (i.e. the name of a procedure—in later chapters we will intruduce more complicatd action patterns). The tutoring rules are scanned to choose one whose Situation part best matches the current state of the world, as represented in the static and and dynamic knowledge bases. (If more than one rule matches, a "conflict resolution strategy" is invoked to select one of them.) Then the Action named in that rule is executed. Some actions may be invisible to the student (we can call these "virtual actions"), such as deciding to change the focus of the discourse (without actually doing it yet), and others will be evident to the student, such as deciding exactly what the new focus will be, and making some utterance toward this new focus. In either case the action will directly or indirectly result in the dynamic information being changed or updated. Then the process is repeated, and typically a new action will be chosen since a different tutoring rule will be chosen to match the information in the updated data base.

3.4 The expert model

It is clear that a tutoring system for some domain, say physics, which can only ask questions about a domain and determine the correctness of the student's answer, is inferior to a system which "knows" physics, i.e. has some representation of physics concepts and problem solving procedures, and can itself solve physics problems. Representing expert knowledge with sufficient

detail and clarity to enable the tutor to be able to solve any problem which it gives to the student (and explain itself as it does so) is quite difficult in most domains. For some domains, the difficulties in building such an expert may outweigh its advantages, and it may be pragmatic to concentrate on the knowledge of how to teach the domain, rather than represent all of the knowledge to be taught. Below are some benefits of including an expert problem solving component:

- The student can ask unexpected questions about the domain, or ask the tutor to solve all or part of a problem.
- Common student errors or misconceptions can be modeled as deviations from the expert knowledge.
- The student's actions can be compared to what the expert would have done. Given a wrong response, the tutor can try to pinpoint which problem solving step or prerequisite knowledge that needs attention.
- An expert system can model expert problem solving behavior for the student to observe and interrogate.

A teacher is more than an expert in her domain (in fact, being an expert may not even be necessary in some cases), she is an expert at communicating the knowledge of the domain. In the next chapter we discuss the importance of including explanatory and pedagogical information in the domain model.

3.5 The student model

If tutoring is an attempt at helping the student "travel" from one knowledge state to another, the tutor must know the student's current "location." Consider an analogy of wanting to transport someone to Fort Worth. If you arrange for them to have a plane ticket from Chicago to Fort Worth, it is of no use if they currently reside in Miami. The analogy to instruction seems obvious, but traditional education (using the "learner as container" model) has often ignored this need to consider the initial knowledge state of students before teaching, and the need to keep abreast of the students' changing beliefs while teaching. Constructing an accurate and/or detailed student model is perhaps the most difficult and important computer tutoring system ability in terms of modeling human tutoring. In the next chapter we will outline the types of knowledge which are needed for a student model.

3.6 Knowledge acquisition

Designers of tutoring systems can have two complementary goals: to design innovative instructional environments/tools which would be impossible without the use of a computer, and to incorporate the knowledge of good human teachers into a system to achieve the level of instructional efficiency possible if the teacher could teach each student individually. Both goals are worthy foci of research. Designing innovative environments is more easily attainable given the state of AI technology. This paper addresses the second goal, modeling human tutoring, which tends to make stronger demands on AI technology. The main theoretical issues are representing knowledge, efficient use of the knowledge in controlling the system's actions, and transferring knowledge from the teacher or expert to a computer knowledge base. This last issue is called knowledge acquisition.

Many different kinds and levels of knowledge are used by a human tutor. The process of codifying this knowledge requires a more precise vocabulary for describing the many facets of tutoring than is now available. Examples of things for which we do not have accepted terminologies for are: types of knowledge, categories of tutoring actions, types of problems (problem solving problems), goals and functions of teaching, and types of examples (examples of concepts, example problem situations, etc.). Terminological precision is necessary because tutors make decisions based on the implicit categories that the properties of the tutoring situation fall into. For example, suppose a tutoring rule says "IF the information being taught is procedural knowledge, THEN ..." There is an implicit assumption that all information to be taught belongs in a "procedural" or a "non-procedural" category; or perhaps the categorization has several groups, such as "procedural", "factual", and "conceptual."

It is preferable from a theoretical perspective for these categorizations to indicate ontological commitments concerning the structure of knowledge and actions themselves. It is acceptable, however, to invent taxonomies that make it easier for the expert to describe his knowledge, and easier for the computer to access its knowledge base. At various points in this paper we will introduce terms for knowledge categories and system functional blocks—ideas for slicing the worlds of knowledge and action with the two goals above in mind. This will make the process of transferring the knowledge from expert teacher to computer knowledge base more efficient, and less ambiguous.

In chapters 4 thru 6 we intruduce a framework for representing declar-

ative and control knowledge which should make knowldge acquisition more tractable. In chapters 7 and 8 we outline a knowledge acquisition methodology for ITS.

Chapter 4

KNOWLEDGE REPRESENTATION

Many diverse types of knowledge are used in tutoring systems. In expert system research much effort is spent on methods and models for representing knowledge. Tutoring systems present even bigger challenges. Each unit of knowledge to be taught (or unit of expert knowledge) must be associated with knowledge of methods for explaining or teaching it, how it relates padagogically to other knowledge, common errors seen when teaching it, and so on. Ultimately, there will be much more of this extra knowledge (which we will call pedagogical knowledge, and could be referred to as meta-knowledge) than expert knowledge in a tutoring system. In addition to all this the tutor needs knowledge about how to tutor and how to diagnose the student.

In this chapter we present a general knowledge representation architecture for tutoring systems. Figure 4.1 illustrates the functional components of the tutor's data base. The static components, on the top row, can be considered permanent knowledge (although this may not technically be the case if such a system is implemented). The bottom row shows the dynamic components, which are updated during the tutoring session. The Decision Mechanism, which matches the tutoring rule conditions against the information in the other data base components (shown as input data), will be discussed in a later chapter. The Action mechanism will also be discussed later.

Below we outline these contents and implementation issues for each of the components of the tutor's data base.

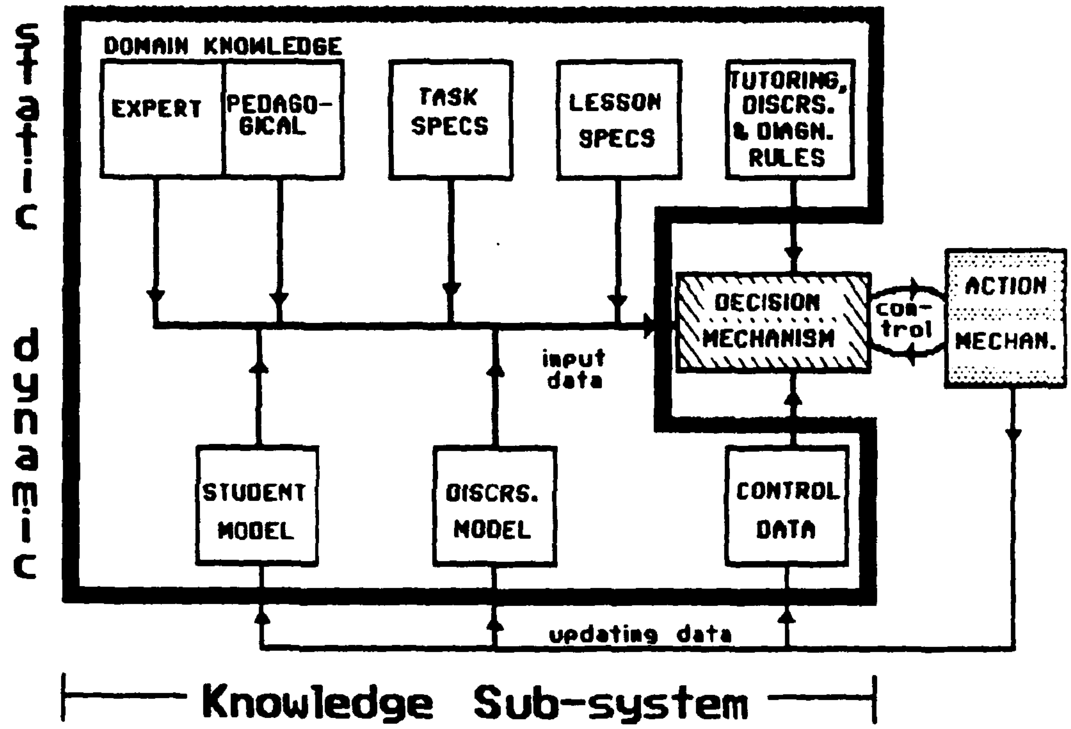


Figure 4.1: The Knowledge Sub-system

4.1 Representing domain knowledge

Some ITS systems have expert systems incorporated into their design. Tutoring can obviously be more effective if the tutor has an "expert's" knowledge of the domain, i.e. an ability to solve problems in the domain. The details of what kinds of domain knowledge needs to be represented, how it is structured, and how it is transferred from a human expert to the computer tutor's knowledge base depend on the peculiarities of the domain, but some general guidelines can be given. Below are some of the important considerations in designing a knowledge representation for a tutoring domain.

4.1.1 Types of knowledge

Different types of knowledge differ in the ways that they are learned and used. For example, memorized facts and problem solving skills differ greatly in how they are learned and used. The efficacy of our tutoring rules will in part depend on the precision and accuracy of our language for categorizing types of knowledge. This assumes that there is some (as yet mostly unknown) useful mapping from types of knowledge to methods for teaching each type (Clancey 86B and Chandrasekaran 85 have some relevant suggestions, but our analysis will be along different lines). A general classification of knowledge into facts, skills, and concepts is given below. Further refinements, motivated by pedagogical factors, will be suggested in a Chapter 8. Appendix B is an example of a more refined taxonomy, showing what such a classification may look like, but which is ad-hoc. (Also, see Murray 86 for more detailed discussions of the categories below in the context of the domain of elementary physics tutoring.)

Facts and skills. The Facts category roughly corresponds to our common conception of discrete factual knowledge. Factual knowledge includes definitions, propositions, properties of things, and relationships between things. Examples of factual knowledge are: the size and color of a ball, George Washington's birthday, the definition of chemical equilibrium, the relative heights of two houses, a mathematical formula, and whether or not two people are neighbors. One can see from this example that it may be useful to subdivide "facts" into sub-categories (but we will not do so).

The Skill category represents our knowledge of how to do things. Skills are procedures, algorithms, guidelines, heuristics, etc., which have a sequencing of physical and/or mental actions associated with them. Examples of

skills are: the long division algorithm, how to construct a geometry proof, and how to buy groceries.

Concepts—Nexus and Deep. Concepts represent groupings of densely interconnected knowledge of diverse types. Concepts can be visualized as nodes in a semantic network of knowledge, or entities representing structural relationships between other pieces of knowledge. The concept of gravity, for example, includes formulas (facts), experimental knowledge about the effect of gravity, skills for measuring the effect of gravity, knowledge of when to apply the associated facts and skills, and intuitive knowledge from experiencing the effects of gravity. For the purpose of representing knowledge in tutoring systems, we will make a distinction between two types of conceptual knowledge: Nexus Concepts, and Deep Concepts. Both Nexus and Deep Concepts are representations in the computer tutor of knowledge which we wish to teach to the student. Nexus concepts are represented completely in the tutor, so that we can say that the computer "understands" the meaning of the concept and can solve problems requiring its use. (By a computer "understanding" the meaning of a concept, I mean nothing more than that it can reason about it.) Deep concepts are those that we expect the student to eventually learn, but which we do not, and/or can not represent with enough computational precision to allow the tutor to solve the problems using the concept. (The term "Deep" here refers to complex, deep understanding in the human, so the computer actually has a "shallow" understanding of Deep Concepts. I'm serching for a better term than "Deep.")

Some of the conceptual knowledge which people use to solve problems is beyond the representational state of the art because of its complexity or lack of definition. Clancey (86, pg. 301) says: "The totality of what people know about a concept usually extends well beyond the schema that are pragmatically encoded in programs for solving limited problems." Even though a Deep concept can not be modeled effectively in an expert system, we may be able to represent sufficient aspects of it to be able to teach it, such as knowledge of how to teach it, examples of uses of the concept, and specifications of behaviors that give evidence that students knows the concept.

Both Nexus and Deep Concepts represent structural relationships between pieces of knowledge. Understanding a Nexus Concept is equivalent to understanding all (or perhaps most) of its components and their relationships which are represented in the computer. Understanding a Deep Concept requires knowing more than the components and relationships explicit in the

computer's rendition of the Deep Concept.

The distinction between these types of knowledge is not made in the literature, probably because there has been little or no intelligent tutoring of Deep concepts (but see Murray, Woolf, et. al. 86 for a recent attempt).

As an extreme example of a Deep concept, consider the concept of friendship. As part of some tutoring session we may want to know whether our student has this concept. We may be able to infer from some testing or behavior that the student has the concept, but we can not represent "friendship" precisely in a computer knowledge base. In contrast, we may be able to specify all aspects of what it means to understand the concept of a FOR loop in PASCAL programming. Such a concept would be a Nexus concept. Whether a concept is of the Nexus or Deep type is not strictly a function of the concept itself, but depends on whether we decide to (or can) capture its meaning with computational precision in our domain knowledge base.

Expert vs. pedagogical knowledge. The domain model is an expert system in the domain with copious information of explanatory or pedagogical nature. Builders of expert system have come to appreciate that there are many circumstances under which it is important that an expert systems not only arrive at an expert's solution, but that it do so in a way which simulates a human expert's problem solving process. It is also now an accepted maxim in expert system design that such systems be able to explain how and why they take the steps they take as they pursue a problem solution. For expert system builders this facilitates knowledge acquisition, debugging, and upgrading of the programs (see Clancey 86A). If computer tutors are to incorporate expert problem solvers, it is even more important that these programs emulate human behavior and can explain their solution steps (Clancey 82, and Brown, Burton, & deKleer 82).

We call "expert knowledge" the knowledge needed to solve a problem in the domain. "Pedagogical knowledge" is additional knowledge needed to explain, justify, or teach the expert knowledge. Knowledge about functionality, purpose, causality, exemplars, prerequisites, etc. are included in pedagogical knowledge, as well as annotations concerning learning difficulty, importance, salience, and suggestions on how best to teach pieces of knowledge. Pedagogical knowledge also includes common erroneous facts, buggy procedures, and misconceptions associated with pieces of knowledge. As mentioned above, a successful tutor is likely to need a high ratio of pedagogical to expert knowledge. In figure 4.1 the expert and pedagogical components are shown as conceptually separate, but both types of informa-

Categories of domain knowledge

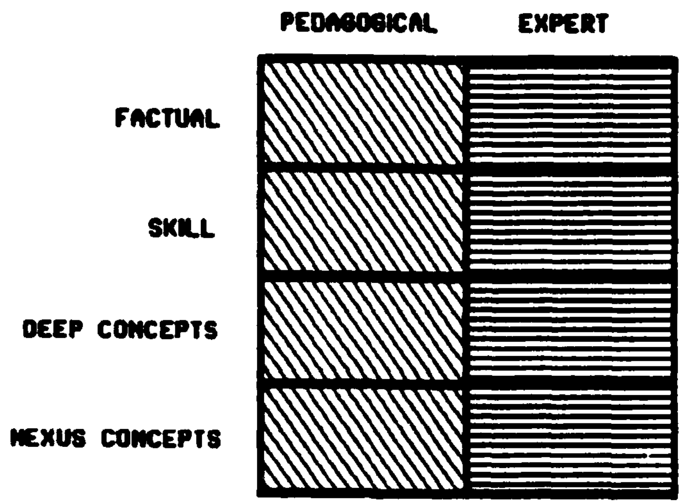


Figure 4.2: Categories of domain knowledge

tion will probably be interleaved in the same representational structure. In fact, it may not always be clear where the boundary between them is, since what constitutes information needed to solve the problem depends on one's definition of an expert problem solution (for example, does it include some explanatory information?). For more reading on "knowledge about knowledge," see Clancey 86B, Rissland 83, diSessa 85, Flavell 80, and Claxton 85.

The distinction between fact, skill, and conceptual knowledge is orthogonal to the distinction between expert and pedagogical knowledge. I.E. there will be some expert and pedagogical component for all knowledge, whether facts, skills, or concepts (see figure 4.2).

4.1.2 The internal structure of domain knowledge bases

When domain knowledge is analyzed for the purpose of teaching, an overall structure for that knowledge is often proposed. The structure is often hierarchical in nature. There are several dimensions over which one can construct such a structure, and there are different reasons for wanting to explicitly outline the structure of domain knowledge. For example, we may at some point in a tutoring session want to specify that "Newton's Third Law" be the next topic, and later another rule may refer to teaching "forces in static equilibrium situations", which is a component of teaching "Newton's Third Law". Here we need representations of different levels of generality. We may also need to represent prerequisite levels, or levels of causality. We have no suggestions here for synthesizing the various ways to structure knowledge and the various uses for these structures, but it seems worthwhile to mention several instances of domain knowledge analysis of structure. We do so below.

Chi et. al. (81) and others have shown that one difference between expert and novice approaches to physics is in how elements of problems are classified. Novices tend to classify according to surface features, and experts have something like a classification hierarchy indexed by non-surface properties. This suggests that both relevant and (to the expert) irrelevant factors must be represented in our expert knowledge base, so that non-optimal student behavior can be recognized and responded to intelligently.

Carbonell & Collins (Carbonell 70), in the SCHOLAR project, found it necessary to distinguish between "super-attribute", "super-part", and "super-concept" links in representing geography knowledge in a semantic network. Super-attributes link objects or concepts with important attributes. Super-concepts show "a kind of" links, such a Peru is a country. An example of a Super-part link is that Peru is a part of South America.

Stevens, Collins & Goldin (82) (the WHY system), and deKleer & Brown (80) have investigated the causal and enabling interconnections between facts and events. WHY's domain, meteorology, contains knowledge about the workings of a physical process. In teaching any domain dealing with explaining the science behind physical mechanisms we may expect that causal interconnections will be important. White & Frederiksen (86), in their electronics tutor, also teach causal relationships between physical mechanisms. To contrast, Clancey's GUIDON (82) shows how information accumulated during diagnostic problem solving can be represented in an evidential network, with each new bit of information pointing back to the piece of infor-

mation which supported its instantiation. Causal analysis of the system, i.e. anatomical processes, is not taken into consideration.

Burton (82), in the BUGGY system, an analysis of subtraction skills, found that the skills could be organized in a "skill lattice": a hierarchical structure representing how subskills use, subsume, or mask other subskills. Barr, Beard & Atkinson, in the BIP tutor for teaching BASIC programming, use a "Curriculum Information Network" to relate tasks to domain skills. The network contained information about which skills were prerequisites of others (dependencies), and contained kind-of and component-of relationships similar to the super-concept and super-part relationships in Scholar.

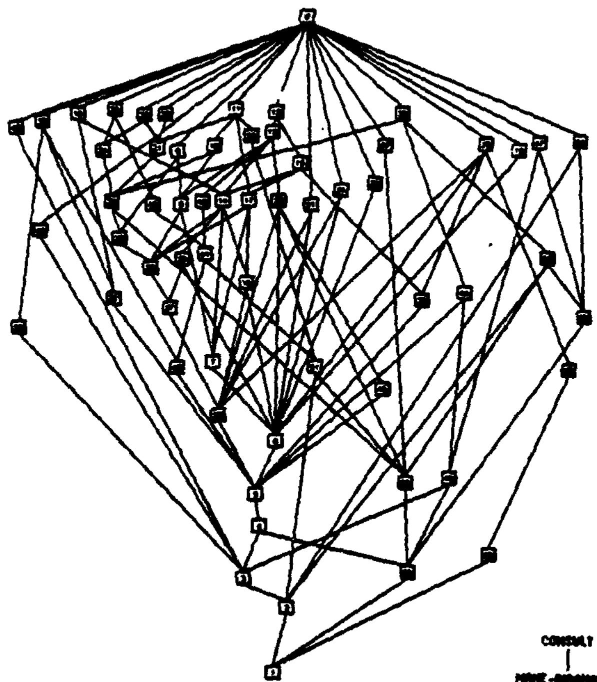
Miller (82), in the Spade-0 system for teaching LOGO programming skills, shows a hierarchical taxonomy for steps in planning or debugging computer programs. Anderson's LISP tutor (Anderson & Reiser 85) and Geneserth's (82) Macsyma Advisor encode problem solving knowledge (or "planning" knowledge) in subgoal structures.

Goldstein's (82) WURSOR tutor represents knowledge as procedural rules which are interrelated by the relationships analogy, refinement, generalization/specialization, and correction/error. It also incorporates phases of successive refinement of domain knowledge, where succeeding phased precludes previous ones, approaching an expert's knowledge.

The above research indicates that the hierarchical or overall classification structure of knowledge is used in various ways. Knowledge of subconcepts, subskills, and prerequisites is used in diagnosing misconceptions and prescribing remedial activities. The same knowledge can be used to direct the presentation of new knowledge in a tutor's default teaching path. Hierarchically structured knowledge allows us to refer to groupings of knowledge in tutoring rules and in tutoring session plans. For example, we can specify to "teach about energy conservation", and the system can infer from the hierarchy what sub-concepts must be taught. It is likely that several of these organizational structures will have to be represented simultaneously (in parallel) in an ITS knowledge base. Utilizing different organizational structures during a tutoring session allows knowledge to be connected in different ways, or approached from different angles.

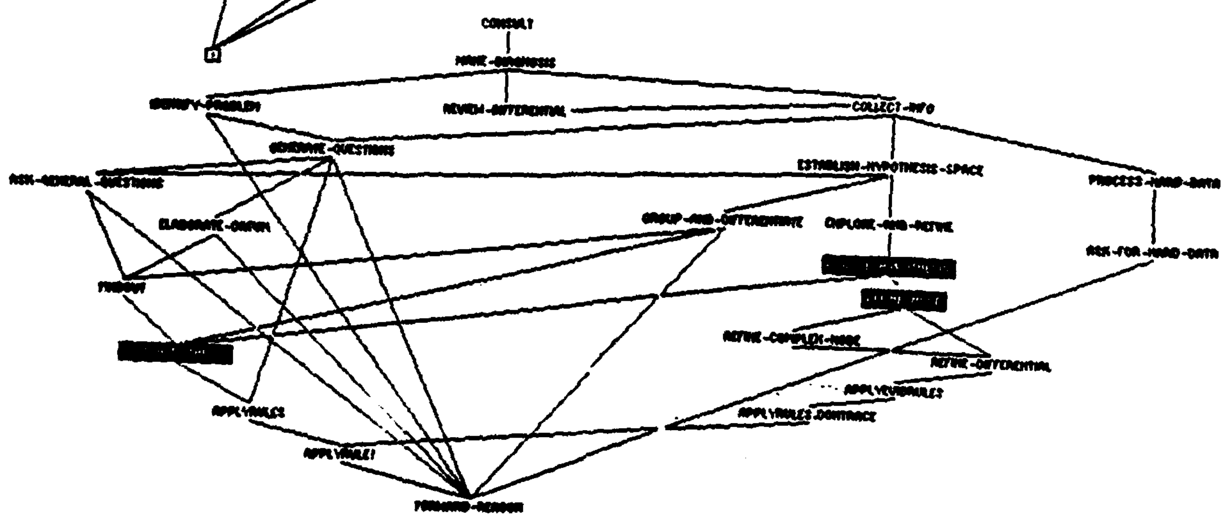
4.1.3 Uses for domain knowledge

We will call each piece of knowledge that we wish to teach a "knowledge unit", or KU (from Servi 86). As indicated in the above discussion on



Skill Lattice for subtraction
(Buggy system, Burton 82)

0 Correct skills
1 No subtraction skills
2 Subtract 0 from a number whose right-most digit is 0
3 Subtract 0 from a number
4 Subtract columns which are n-n or n-0
.
.
.



Lattice for Diagnostic Task Invocation
(Neomycin system, Clancey 86)

Figure 4.3: Example knowledge base structures

knowledge structures, some KU's may subsume lower level KU's. KU's may represent a single fact, a single rule, a single step in a procedure, or complicated concept or skill. The most important use of KU's is in representing expert knowledge. Expert knowledge is emulated in order to show the student the knowledge, to allow a simulation to exhibit that knowledge, or to check a student's behavior against what an expert would do.

KU's can have several attributes associated with them besides those which allow expert emulation. Examples of pedagogical information that can be associated with each KU are: how to give examples of itself (see Rissland, Valcarce, & Ashley 84, Lenat 79, and Murray et. al. 86); describing its purpose, i.e. why know it or why use it; explaining when it should be used, i.e. under what problem solving context; the ability to quiz the student to determine if the student knows it; listing its prerequisite KU's; listing the assumptions implicit in the KU; and revealing the source of the KU knowledge, or an explanation/justification of how it was derived or inferred.

4.1.4 Object-oriented representations

We are using an object-oriented representation of the domain knowledge, which affords several desirable characteristics (see Bonar 86, and Stefik & Bobrow 85). In object oriented programming, data and procedures are combined into entities called "objects." An object is a data type combined with procedures (or operators) called "methods" specific to that data type. In this programming paradigm, one "sends a message" to the object, rather than calling a procedure. In our architecture, Knowledge Units, Examples, Lesson Specifications, etc., are all types of objects. Each type of object has methods associated with it for the operations specific to it. For instance, different types of knowledge (.i.e. sub-types of the Knowledge Unit type) may have different procedures for evaluating misconceptions related to them. One advantage of this formalism is "data-abstraction," meaning that the use of a procedure associated with an object does not require that the user know anything about the internal representation of the object (see Bonar, Cunningham, & Schultz 86). Without object oriented programming we may have to specify different diagnostic procedures (Diagnose-1, Diagnose-2, etc.) for each type of knowledge. Using methods, we send the message "(SEND knowledge-unit DIAGNOSE)", and the object receiving the message (the knowledge unit) will have its own diagnostic procedure executed. KU's can have methods such as "teach-me", "describe-me," "test-for-me," and "tell-prerequisites." A tutoring rule can specify that an example of the current

concept be given to the student, and the fact that different types of concepts must use different procedures to give examples of themselves is taken care of automatically.

Object oriented programming environments also have a mechanism for hierarchical relationships, facilitates classification grouping and property inheritance. For example, the general class "KU" may have a method for teaching its prerequisites. KU's may be in a hierarchy of categories of knowldge, such as Facts, Skils, and Concepts. These categories will, by default, inherit the "teach-prerequisites" method from the KU type. If, for example, the way Factual knowledge teaches its prerequistes is different than the norm (the default), we can define a new "teach-prerequisites" method associted with Facts.

4.1.5 Mental models and qualitative reasoning

There has been much focus of late on qualitative reasoning processes and understanding the internal mental representations people have of systems (Gentner & Stevens Eds. 83). It appears that expert problem solvers in many domains are able to picture, "run", or "envision" the workings of a system using imagination alone. Given the initial conditions of the system, the problem solver can "run" the model and make predictions about its behavior. Furthermore, poor problem solving abilities can sometimes be attributed to incorrect mental models (White & Frederiksen 86). Mental models deal almost exclusively with qualitative knowledge. In fact, it was research on the importance of qualitative reasoning which lead to the current interest in mental models. Several studies have tried to explain why students who could solve routine quantitative problems and do well on academic exams in math and science course have much difficulty with nonroutine problems and problems which required assumptions and estimations (Lockhead & Clement, Eds. 86). It was found that deficiencies in qualitative understanding of these domains was a factor.

In tutoring systems, especially in technical domains, the importance of qualitative reasoning and qualitative questions should be stressed. Unfortunately, given the present state of the art, it is difficult to build computer expert qualitative problem solvers (but see deKleer and Brown 82, Forbus 84). This is an instance where the "Deep concepts" mentioned above may be needed. We want to teach qualitative knowledge, therefore we would like to be able to refer to instances of this knowledge (KU's) in our student model and our tutoring rules. The resulting representations for these Deep

concepts merely "stand for" the concepts, and can not be used in an expert system to solve qualitative problems. Several existing tutoring systems emphasize qualitative knowledge of physical systems and attempt to help students build correct and robust mental models (Burton & Brown's (82) SOPHIE, and Holland's STEAMER).

4.1.6 Novice domain models and moving targets

Two related issues will be discussed here. One is that an expert's representation of the knowledge in a domain may not be cognitively accessible by the novice in the domain. The expert has arrived at an efficient and powerful structuring for the many pieces of knowledge that comprise his expertise. It may not be possible to teach this expertise directly in this "compiled" form; attempts at teaching it may all be "over the student's head". In such a case it is necessary to incorporate knowledge of how the expert built up this knowledge from a novice state. Intermediate "non-expert" representations of the domain knowledge may be needed.

The second, related, issue is that the student model is an evolving set of knowledge (and mis-knowledge) pieces. In order to diagnose student knowledge we may need to represent successive levels of evolution from novice to expert understanding. Goldstein's (82) genetic graph is an attempt at representing the evolutionary connections between KU's. His Wumpus tutor can focus its remedial interventions around an appropriate level of expertise. Burton & Brown's WEST research (82) mentions adjusting the level of play of a computer coach if the student loses consistently.

White and Frederiksen (86) discuss "model progression" in student's understanding, and teaches successively more refined models of electric circuits. Anderson (83), in his ACT theory of cognition, proposes mechanisms for "knowledge compilation", and suggests designing tutors which can adjust their model of the student when separate pieces of knowledge are compiled into efficient units.

4.2 Task and diagnosis specifications

In our proposed architecture we separate knowledge about specific tasks from general (expert and pedagogical) domain knowledge because it is used and represented differently. Tasks involve applications of domain knowledge to specific situations.

4.2.1 Tasks

Tasks are problems for the student to solve, presentations with examples and questions to be given to the student, or other activities for the student to observe or engage in. The tutor may give a student a task as a means of teaching something, or as a means of diagnosing the students knowledge, or both. Task specifications describe how the task is to be set up, and how to interpret the student's behavior while she is engaged in the task. The simplest example of a task specification is a question (a block of text) and a description of correct answers to the question. An example of a more complicated task specification is initial conditions for setting up an electronic circuit simulation, complete with methods for analyzing the student's behavior relative to an expert's behavior. Task specifications can be arbitrarily underconstrained, expecting many details to come from the current tutoring context. For example, a task specification may set up a medical diagnostic case such as in GUIDON, and look at the current tutoring context for what disease the simulated diagnosis will diagnose.

4.2.2 Diagnostic specifications

The task specification can also contain information about how student behavior gives evidence for (or against) the existence of concepts or misconceptions in the student model. We have noticed, in designing tasks and rules for tutoring systems, that knowledge of how to diagnose a student's behavior is most often specific to a particular task (as opposed to a general diagnostic strategy). As a simple example: if the student answers "a" then increase the confidence that she has concept X, and if she says "b" then increase the confidence that she has misconception Y, or if she measures item C more than 3 times, she is missing concept Z.

For diagnostic information that are general to many tasks would be stored in the diagnostic tutroing rules (as opposed to in the diagnostic specification of a task).

Recall that diagnostic information can also be stored as pedagogical information in the domain model, provided it refers to a specific piece of knowledge. For example, associated with a concept we might find a pointer to a task that tests for that concept, or a more general description of behaviors that give evidence for (or against) that concept. So a task may point to a concept, indicating that the task requires knowledge of that concept, and a concept may point to a task, indicating how certain behavior in the task is relevant to that concept.

4.2.3 Simulation modules

We will assume that simulation modules and interface modules that can interpret the student's raw behavior (such as mouse clicks, text typed in, and operations performed to manipulate a simulation environment) exist outside the tutoring system architecture described here. The task specifications send information to these modules in a language understood by the specific module, and the module communicates with the tutor in a language designed for exchanging information with the student model, the domain model, and the other parts of the tutor. Note that this architecture assumes that a simulation is a component of the tutor that was designed after or along with the pedagogical aspects of the tutor. This is to contrast with designing a simulation first and trying to tuck the pieces of a tutor in around it.

4.2.4 Tasks as specific actions and instantiations of knowledge units

One can think of the description of the task as an instantiation of knowledge in the domain expert knowledge base. A task description may say that Frank throws a 5 lb. ball into the air with an initial velocity of 10 m/sec, and then ask several questions about this situation. A physics expert knowledge base knows about projectiles, gravity, and how to answer questions such as the one given, but it does not know about any particular situations, such as the one with Frank and his ball. Particular problem situations are specified in the task specification component. Our architecture also has components called Tutoring Actions (discussed in a later chapter). Tasks can also be thought of as instantiations of tutoring actions, or as specifications of general tutoring actions. For instance, a tutoring action may be "give an example," or "start up the simulation," and a corresponding task would specify the specific example given or parameters for the simulation.

4.3 Tutoring rules

Tutoring rules encode general knowledge about tutoring, communication (discourse), and student diagnosis. They incorporate information about what to do (for example: give an extreme case example, diagnose a misconception, etc.), when to do it (for example: whether to ask the student a question now, or not interrupt the student yet), and how to do it (for example: what form an utterance will take). For the time being, we can

picture the tutoring rules as production rules, i.e. as "IF...THEN" statements, where the IF part specifies some state of the world, and the THEN part specifies some action to take if the current context matches with the IF part. (Note: In chapter 6 we introduce some modifications to this basic rule formalism.)

Tutoring rules may contain information about how to teach or diagnose knowledge in a domain, but, in contrast to the pedagogical knowledge in the domain KB (knowledge base) and the diagnostic knowledge in the task KB, tutoring rules are fairly general. For instance, a tutoring rule may describe a state of the discourse where "giving an example of the correct concept" is appropriate. The pedagogical information in the domain model would know what the examples are for each concept.

In most ITS's the tutoring rules are not explicit, or are reported in vague or general terms. As such, they are more like general tactics than computationally precise rules. In this paper we advocate incorporating tutoring rules explicitly as production rules (or related structures as discussed in Chapter 6), which can be examined and modified by the designer. (See Woolf 84 for a theory of organizing the set of tutoring rules in an ITS in a network structure.) Tutoring rules will be discussed in more detail in Chapter 6.

4.4 The student model

Here we will discuss the types of knowledge needed to model the student's beliefs. For discussions on the complex tasks of diagnosing and modeling the student, see Clancey 86B, Anderson, Farrell, & Sauers 84, VanLehn 86, and Burton 82.

The student model usually contains an "overlay" (Goldstein 82) of the expert components of the domain model, i.e. a copy of (or pointer to) the KU's of the domain. These pointers can be annotated with information about the estimated strength of that knowledge in the student, and the uncertainty or reliability of the system's belief that the student has that knowledge. An overlay student model might contain information such as how often the KU was used, how often the student asked for help concerning that KU, whether the major prerequisites of that KU are met, etc. Note that some of this information is technically not a model of the student's knowledge, but evidential data supporting components of the student model.

The relationship between expert KU's and mis-knowledge units in the student model depends on the type of knowledge (whether fact, concept,

skill, etc.). For example, facts which are properties of things can be missing from the student model, or can exist with the wrong value. Representations of erroneous Deep Concepts, called misconceptions, may be pre-defined in the tutor, based on research on common misconceptions in a domain. Skill or procedural knowledge has several mis-skill variations. A skill can be applied correctly but in the wrong context, or it can be missing when needed, or it can be applied when needed but not executed correctly.

Much was said in previous sections concerning the student model and student diagnosis. To summarize: the student model may be a moving target, evolving as the session progresses; sub-expert levels of knowledge and "genetic links" may be needed to model student's knowledge; diagnostic specifications may be associated with certain tasks, and diagnostic actions may be pointed to from individual KU's; hierarchical organization of expei, knowledge can indicate possible areas of weak subconcepts or subskills.

As mentioned earlier, expert knowledge may in some cases be too advanced for students to assimilate, and a representaiton of "novice" knowldge may be needed. Novice knowledge, though not as general or precise as expert knowledge, is more accessible. It can be used as an instructional stepping stone, or for more precision in diagnosis (i.e. an overlay of the novice knowldge units along with the expert KUs).

One advantage of representing the student model as a subset of the expert knowledge (along with deviations from expert knowledge) is that the student model can be "run" just like the expert model can. In so doing we can make predictions, using our current model of the student, about how she will behave given some task. These predictions can be used to generate appropriate task parameters, or to check the validity of the student model by comparing the prediction with the actual student response.

The student model can also contain more global descriptors of the student's cognitive or affective state (i.e. not tied to specific KU's in the expert model). For example: learning preferences, methods of tutoring that worked well for this student, general confidence level, etc.

4.5 The discourse model

The discourse model is a dynamic representation of the current state of the tutorial discourse, including information about what actions have transpired during this (and perhaps previous) tutoring sessions. Compared to the student model, which is a representation of what the student *knows*, the dis-

course model represents what the student and the tutor have *done*. Perhaps more important than a historical listing of significant events, are summary or abstracted properties such as the total number of wrong answers in the session, the number of changes in focus, how long ago the student asked for help, etc. This summary information can be updated continuously, or the history can be scanned backwards from the present to infer this information when it is needed.

The discourse model remembers the context in which the actions it records, or the behaviors observed, occur. For example, in Shute & Bonar (86) behaviors are recorded for eventual matching with test conditions "are all the simulation variables selected for recording in the student's notebook only those actively changing?"

The vast majority of ITS's incorporate what we are calling the discourse model into the student model, or have have no discourse model information at all.

4.6 The control data base

The control data base holds information, mainly for "bookkeeping" purposes, which is used by the decision mechanism, and which is not approprite for the Student or Discourse Models. Among other things it allows for goal driven dialogues, nested sub-goals, and schedueling of tutorial activities. It will be discussed further in the chapter 6.

In summary, we have indicated many ways to slice up the knowledge needed by an intelligent tutor. Since this framework has not yet been fully implemented, it is only a first pass suggestion. Ultimately, there are two criteria for deciding whether to make any given slice through the knowledge, i.e. whether to define a given distinction between types of knowledge. The first criterion is that the distinction must be referred to in the tutoring rules (or in forseen tutoring rules). That is, each new category is introduced because the designer encounters a tutoring tactic which distinguishes some characteristics of the knowledge (i.e. a distinction should not be incorporated just because it is philosophically appealing to think of the knowledge as falling into certain categories). For example, assume we have a category of knowledge called "Rules." It is observed that an expert teacher interrupts a student when he makes a mistake using "memorized" rules but not when the student makes mistakes on "heuristic" rules. In order to incorporate

this new rule into a system, we should subdivide the "Rules" category into "Memorized Rules" and "Heuristic Rules." The second criterion is that the distinctions made are defined precisely enough to be useful. for example, we will need a clear definition of the difference between the Memorized and Heuristic rules. As onther example, suppose that we find that in some domains it is practically impossible to judge whether a piece of knowledge is a fact or concept. In such a case it does no good to have a place to record this distinction in the knowledge base, or to refer to this distinction in tutoring rules.

4.7 The lesson specification

The lesson specification is a top level lesson plan, and/or a specification of the high level learning goals for the tutoring session. In contrast with the tutoring rules, which control the moment to moment local decisions of what to do next during a tutoring session, the lesson spec outlines the sequence of main sub- topics, activities, and tutoring styles to be used for a given session or topic unit. The Lesson Spec will be discussed in more detail in the next chapter.

Chapter 5

LESSON–LEVEL INSTRUCTIONAL SPECIFICATION

5.1 Global and local planning in ITS's

The sequence of information presented by a computer tutor can be thought of as being controlled at two levels: a global level and a local level. The global level is the Lesson Specification, where the instructional designer specifies curriculum level lesson structures and topics to be learned. At the local level, decisions are made on the fly by the tutoring rules, regarding such things as what examples to give, whether to present prerequisite materials, etc.

The Lesson Spec includes a top level specification of the instructional goals. Tutoring systems which are controlled only by local decisions made by small grain size (i.e. very specific) rules may wander in their presentation of material because the tutor is "near-sighted," i.e. has no overall goal or plan. A high level (i.e. lesson level) goal specification enables a more focused and consistent sequencing of main topics. The Lesson Spec not only specifies the general content of units of instruction, but also an instructional environment and pedagogical style, as we will outline below. Having such information in the Lesson Specs enables multiple instructional environments and teaching styles in a single tutoring session.

5.2 Learning and teaching at different levels

One factor that adds to the difficulty of computationalizing the human tutoring process is that it is often not simply a one directional path through a knowledge space; rather, the tutor switches focus between local and global issues, giving perspective and then coming back to the details. Also, effective learning in many domains requires some kind of spiraling through the knowledge space, going over the same material several times with different perspectives. It is difficult to evaluate the success of many existing computer tutors because they do not allow for various modes of instruction and multiple passes though material. In most domains one can not expect significant learning to take place with one pass. For example, doing effective problem solving requires knowledge of basic factual knowledge, yet understanding of the factual knowledge can only be obtained in a problem solving environment (or some other rich or realistic context)–a chicken-egg problem. The solution often applied in the classroom or text book is a layered approach in which topics are gone over in several passes, with increasing detail or increasing focus on application.

In a traditional science course a student may first skim the chapter or read the introduction to get an overview. Then she reads the chapter and/or goes to a lecture session to get introduced to the details of the new concepts. Then she engages in homework, labs, and discussion sessions to construct a more intimate grasp of the material, including interrelations between concepts and meta-knowledge about the material. Finally the student tries to put the instructional unit all together and synthesize it with previous material by writing up lab experiments and studying for exams. Computer tutors should likewise allow for several passes while learning a given topic, and have different leaning environments available for the different passes. Determining focus shifts at the global level can be done in the Lesson Spec. Each pass through the material may require a different learning environment and different tasks, and a different set of tutoring rules will be appropriate. Note that the more intelligent our tutors become, the more they will be able to infer things at the curriculum level. The extreme case is a system which knows only the domain knowledge and the pedagogical characteristics of that knowledge, and can decide on its own what the sequence of topics and the appropriate tutoring modes are. However, for the near future we should design systems that allow instructional specialists in the domain to specify some of the global goal structure and tutoring modes.

5.3 Tutoring Modes

One of the popular issues in ICAI research centers around how much control the student should have over his learning environment, and, conversely, how much control the tutor should have (see Papert 80 and Pea 83). The optimal degree of control is a function of many things, including the learning task and the type of "pass" through the learning material, i.e. whether the knowledge is being introduced or used to solve problems. The real question is not which strategies or instructional paradigm to use, but when to use each. This is especially true in trying to design a computer tutoring system which will have the flexibility to accommodate many domains and the instructional styles of many instructional experts. A spectrum of student control from exclusive student control to no student control is outlined below:

- Passive environments which embody the knowledge or properties of the domain; the student has full control in experimentation and play within some environment.

- "Coaching" environments which observe the student's actions, and interrupt only when a mistake is made or a significant learning opportunity is missed.

- "Mixed initiative" environments where the tutor has a definite agenda concerning what is to be learned, but the student has the ability to interact to gather information or change the direction of presentation. (This may be similar to "Socratic" teaching, but interpretations of the Socratic method vary in the literature, so we don't use it here.)

- "Pedantic" environments in which the student has little control, and the tutor presents material in a fairly lock-step fashion, similar to a lecture or a programmed learning environment.

Styles of learning environments such as those outlined above vary along several dimensions, including: 1. the amount of information needed about the student (i.e. the complexity of the student model and diagnosis procedure), 2. the degree of control which the tutor assumes, and 3. the types of facilities which are available to the student for gathering information, changing the direction of the instruction, and changing the parameters or components of a simulation environment.

We propose a data structure called a "Tutoring Mode" (or just Mode) which specifies the pedagogical characteristics of the learning environment

by specifying a set of tutoring rules to be invoked and a set of facilities or "Tools" available to the student. The set of tutoring rules which comprises any mode will usually overlap with the set of rules in other modes, i.e. some rules will be applicable to several modes. Examples of what is meant by "tools" above are commands such as glossary, help, review, new topic, and facilities such as meters, constructors, notebooks, etc. (and see Shute & Bonar 86, and Cook 83). Following is a possible set of tutoring modes, with their intended purposes and functions, which an ITS system may have available to it. They are listed in an order suggesting the "several passes" approach to learning mentioned above:

Summary/preview/or review

Highlight the learning objectives, usually in a context relating it to other topic areas. The student can ask for list of prerequisites and examples of the learning goals. The student can request questions which test competence of the learning objectives. This mode could be used as an introduction or conclusion/summary to an instructional unit, or it could be used as a review.

Pedantic

Here the tutor "lectures" on a topic, or explains, or creates a situation and solves it, giving explanations as it goes (as in the expert trouble shooter of SOPHIE II (Brown, Burton, & deKleer 82)). The student watches, and can interrupt to ask for a more detailed explanation of something, or to ask a specific question about a value, a glossary item, or a "what if" question, altering one of the situation parameters. This mode, similar to what is usually termed a "tutorial", is a bottom up teaching style; new information is introduced and built upon in a systematic way (compare with "mixed initiative" below, which is top down). Some uses of the Socratic teaching method is pedantic, since the student's learning is directed by questions asked by the tutor.

Passive learning environment

The student is allowed to experiment and play without interruption in a rich environment which embodies or simulates the objects, relationships, rules, and/or structure of the domain. No intentional feedback or diagnosis is given. Students can learn from their mistakes provided they can recognize mistakes and infer the cause of the mistake. Since there is no guidance, the environment must be motivationally stimulating. The student can ask questions about values or glossary

information. If the domain is highly information oriented, the student can browse through a data base. Examples of passive learning environments are the LOGO programming environment (Papert 80), and Schwarzt's Geometric Suppopser. Simulations without student feedback are also passive learning environments.

Mixed initiative teaching

The student is allowed to construct his own situation, or he is provided with one. His task is to solve a problem. The tutor has a quasi-rigid specification of what it wants the student to learn, though it may be deciding dynamically how best to tutor this information. The tutor monitors each step closely and provides feedback, diagnostic questions, or hints where appropriate (care must be taken not to interrupt too often). The tutor tries to correct misconceptions. The student is allowed to access a glossary, a calculator, an information "browser", and other tools. She is allowed to ask "what if" questions in a limited way (if they are not irrelevant to the current topic). This mode represents "top down" tutoring. Familiarity (though not mastery) with the basic facts, skills, and concepts is assumed. The tutor starts by presenting a non- trivial problem solving situation, and prerequisite knowledge is taught only as it is needed.

Coaching mode

This mode is appropriate for "informal learning environments" (Brown, Burton, & deKleer 82), such as games and on- the-job training. Burton and Brown also call it a "reactive" learning environment, because the student learns from her mistakes. The tutor analyses student work in the background, interrupting only the student makes a sufficiently serious mistake or misses a significant learning opportunity. Student diagnosis, if it is done, is difficult in coaching modes, since the tutor must be an expert problem solver in the domain, and recognize the student's knowledge, misconceptions, and plans without interrogating her. Coaching has been implemented with (Burton & Brown 82) and without (Brown, Burton, & deKleer 82) student modeling.

Test-out

The student is given a problem and is given no assistance. He may have limited access to tools such as a glossary or equation solver. Moving on to the next topic requires success with the test-out.

Lesson Specification components:

- Tutoring Mode
- instructional goals
- simulation or physical environment
- tools or features avail. to student
- prerequisite knowledge list

Figure 5.1: Lesson specification components

5.4 The Lesson Specification

The Lesson Spec specifies a sequence of main Topic Units (or phases of teaching a topic). Each topic unit contains the following information (as shown in figure 5):

- Specification of the "physical environment": This may set up a simulation, or a particular arrangement of screen windows, etc.
- Specification of the tools or features available to the student: measuring tools, help functions, information access functions, graphics tools. These tools partially determine the degree of control the student has over his environment.
- Prerequisites for this topic: Assumed knowledge and perhaps a pretest.

- Instructional goals: A list of learning objectives for the student to inspect (like a road map of what is expected), a set or sequence of domain knowledge units to be learned, key examples to be presented or problems to be solved.

- The Tutoring Mode: identifies a subset of the available tutoring rules which is applicable to this topic unit. As we will see in the next chapter, narrowing down the set of applicable tutoring rules improves the process of searching for rules which match the current situation.

  Here is an example of a Lesson Specification (its contents are not intended to be realistic):

```
LESSON 5: Pressure in Liquids

     Goals: know liquid-pressure concept; familiarize with
            the boiler-tank simulation
     Prerequisites: area & volume, force

     Topic 1: Area & volume
           Mode: review
           Facilities: glossary, knowledge browser

     Topic 2: Force
           Mode: review
           Facilities: glossary, knowledge browser

     Topic 3: pressure-in-liquids
           Mode: preview
           Facilities: glossary

     Topic 4: pressure-in-liquids
           Mode: mixed-initiative
           Environment: boiler-tank-simulation
           Facilities: glossary, force-meter,
                       teach-in-more-detail-button

     Topic 5: pressure-in-liquids
           Mode: pedantic
```

```
     Give-examples: molecular-interpretation-of-pressure
     Facilities: glossary

Topic 6: Review pressure-in-liquids
     Mode: wrap-up
     Facilities: glossary, knowledge-base browser
```

Before starting Lesson 5 the tutor would check that the student has covered the prerequisites (or if that information is not available, give a pretest). The lesson goals may be just a note for the instructional designer and student to see why the lesson is being given, or could be used by tutoring rules which use the information to make decisions. There are five Topic Units in this Lesson Spec. The first two review previous concepts. The next three teach the main concept, pressure in liquids, from different perspectives. First a preview of the topic is given, then there is an opportunity to explore the concept and take measurements in a simulation environment, and lastly there is a guided discussion on the topic. Note the specification to use the molecular-interpretation- of-pressure. This shows how the instructor can require specific tutoring actions from this global level. Most examples given to the student will be presented on the fly as a result of specific tutoring rules which draw upon information in the Domain and Task knowledge bases.

To summarize, tutoring rules combined with pedagogical information in the data bases dynamically determine tutoring actions at a local level, and the Lesson Specification is used to organize tutoring from a global level, making certain tutoring actions mandatory, and restricting the set of tutoring rules applicable at a given time.

Chapter 6

A GENERAL CONTROL ARCHITECTURE

6.1 Program control issues

The main issues in program control are 1. how information is stored (represented), retrieved, and passed between component parts of the program, and 2. how the sequence of actions the program will take is determined. Previous chapters have outlined structure for representing the knowledge needed in an ITS system. In this chapter we will introduce a control architecture which directs the actions that the tutor takes based on the contents of the these knowledge bases. This architecture is an attempt to systematize the actions and functions of existing tutoring systems.

The proposed architecture is a rule based system with several embellishments. Figure 6.1 shows an abstract view of the flow of information and control to and from a "Decision Mechanism." Note that figure 6.1 includes all the parts in figure 4.1 (the data bases) in a different conceptual structure, emphasizing control rather than knowledge representation issues. (In figure 6.1 the tutoring rules and control data base are depicted separately from the rest of the static and dynamic data bases in figure 4.1.) The tutoring rules and the control data base are considered a part of the "Control Sub-system." The Control Sub-system utilizes information from the "Knowledge Sub- system" in determining which tutoring action to execute next. The Control Sub-system then tells the "Action Sub-system" which action should be executed. The Action Sub-system then executes the specified tutoring action, along with other actions which maintain the environment and data

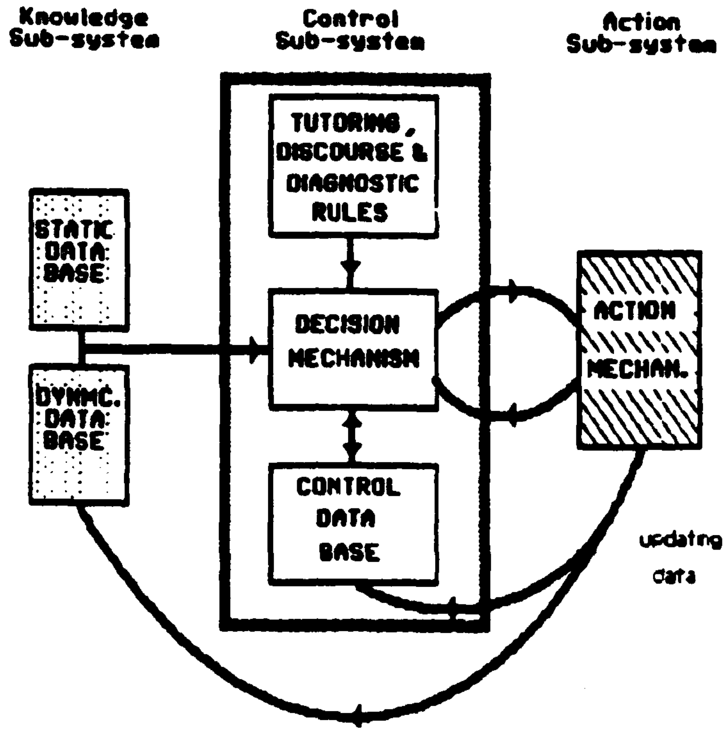


Figure 6.1: The flow of information and control

bases. Finally, control is returned to the Control Sub-system so that it can select the next action. Thus control alternates back and forth between the Control and Action Sub- systems.

Below we will discuss how the Control Sub-system is an embellishment on standard rule based control architectures (which we will also call "simple" or "vanilla" rule based architectures). Then we will expand upon what happens in the Action Sub-system. (Note: The next two sections can be skipped by those familiar with AI production system architectures.)

6.2 Vanilla rule based control structures

Control in a rule based system (also called a "production system") is accomplished by matching "IF..THEN" rules to a data base of information. The rules have the form "IF Situation THEN Action", where the Situation

usually contains several conditions (also called "antecedents") which must all be true for the rule to apply. The Decision Mechanism in a simple rule based system has a "matcher" that checks all of the rules and determines which ones apply (i.e. it looks for a match between the rule's antecedents and the information in the data base). It then chooses one of the rules that was successfully matched, and executes its Action (or "fires"). An algorithm for choosing which of several equally matching candidates to fire, called a "conflict resolution strategy," is needed. The optimal algorithm varies among applications. Examples are: choosing the first rule that was found to match, and choosing the rules whose antecedents are most specific (for example: "if its a dog" is more specific than "if its an animal"). After the rule fires, the state of the world has changed, a fact usually reflected by a modification to the data base. The matching procedure starts again, and this time a different rule should fire since the change in the data base causes different rules to match it. Simply stated, the match-fire process is repeated until the program terminates.

6.3 Advantages and limitations of vanilla rule based control

6.3.1 Advantages

Because production rules have a simple, standard (or "canonical") form, rule based systems are modular, flexible, easily modified, and their actions are easily traced. These are desirable characteristics for our general tutoring architecture, since a standard rule format will allow rules used in a tutoring system to be easily modified and easily transported to other systems (see Davis, Buchanan, & Shortlife 77). As mentioned previously, it is essential that we can easily experiment with and modify tutoring rules.

Rule based systems also have an "opportunistic" flavor to them, in that the flow of control is not rigid, as it is in programs where control is determined by procedures calling each other and passing parameters. Production systems match the rule base with knowledge about the current circumstance after every action. The system continually has its entire repertoire of rules available to it, so it is always ready to respond to changing situations. In tutoring systems this opportunism enables the tutor to react to unpredictable student behaviors.

Vanilla rule based control architectures have several disadvantages though, and we note these below. These limitations become prohibitive as the rule

base becomes large. Vanilla rule based systems with large numbers of rules (or even a moderate number, say 50) are, for reasons given below, hard to manage. These disadvantages are discussed to motivate the embellishments to the simple rule architecture, which we will present later.

6.3.2 Confounding of different types of control information

Because of the simplicity of the control mechanism in vanilla rule systems, different types of control information are treated identically. In complex computer programs this can lead to an undifferentiated and baroque conglomeration of rules.

As an example, suppose that in some tutoring system an analysis of a student's answer can lead to suspecting a student misconception. The tutor checks the student model for previous evidence of that misconception in an effort to confirm its suspicion. If this hypothetical tutor does not find a "high" level of support it increases its level of suspicion, but otherwise performs no visible action. Let's say that we, as system designers, want to add a new rule. If there is "medium" level of evidence, the tutor should ask the student directly if he thinks he believes the misconception. We want the new rule to say "IF there is medium evidence for the suspected misconception, THEN ask the student about it directly". However, to ensure that the rule fires only when we want it to, the rule may have to say "IF the context is one of trying to verify a suspected misconception, AND we just checked the student model, AND there was medium evidence for the suspected misconception, THEN ask the student directly if he thinks he believes the misconception." There will be many rules which are intended to fire only in specific contexts, yet they will be checked for a match after every tutoring action, along with the entire rules base. In such cases antecedents like the first two "ANDs" in the above example rule have to be put in the rules to ensure that the system exhibits the type of structured program flow we get with programming languages, where fixed sequences of actions are easily specified. It is desirable to separate such "control knowledge," from the knowledge about how to tutor (see Lesser 84, Clancey 86A).

6.3.3 "Side effects" are used to control program flow

The action parts of rules are either propositions, as in "IF X is true THEN Y is true", or actions, as in "IF X is true THEN do D". Concerns for clarity would dictate that all the rule actions refer to domain related propositions or actions. But often there are rules, or consequences of asserting the rules,

which exist only to manage program control. For example, if we used production rules to manage the control in figure 6.1, one might be "IF the Decision Mechanism has decided what action to take, THEN run the Action Mechanism." This type of information should not be put in the same rule set with rules which quide tutoring strategies. Also, rules which on the surface are about domain information may have additional "side effects" which set internal system parameters, such as what the last action was, how many times a rule was fired, etc. Tracing the flow of information in the system is more difficult when the control and domain information is treated identically.

6.3.4 Rule structure is opaque

Another difficulty with using rule based systems is that a uniform syntax for all rules hides any structure which they implicitly have. It may be that for some situations the desired flow of control is a rigid "flow chart" like network of actions. Conditionals, looping, and "case" statements in traditional programming languages make this structure explicit. In rule based systems the antecedents have to be rigged, as in the example above, to produce the desired effect, and the underlying structure is hidden to those who need to understand or modify the set of rules.

6.3.5 Lack of focus

Since all of the rules of a vanilla rule system are at the same level of importance, the rules specify primarily low level actions, and decisions are made on a local scale. The system is only concerned with the very next thing it should do. As a result, it is hard to make programs proceed with a high-level focus, and they tend to wander and meander toward their solutions, as if they were navigating with very limited visibility. (Some production systems, such as OPS-5, have been designed to take goal driven control into account.)

6.4 Modifications to vanilla rule based architectures

To eliminate the problems and limitations mentioned above, four modifications to the vanilla rule based architecture will be presented:

- The grouping of rules into functional units (Tutoring Modes)
- An object-oriented data representation for rules
- A restriction on side effects
- A network formalism for representing patterns of rules and actions

6.4.1 Tutoring modes

We have already mentioned one technique which will help somewhat with the above problems. A Tutoring Mode, which is defined as a subset of the tutoring rules, can be specified in the Lesson Specification. This can greatly reduce the space of rules which must be searched during each control cycle.

6.4.2 Frame and object oriented representations of tutoring rules

Representing rules in frame-like structures provides two advantages: rules can be grouped into hierarchical classifications, and rules can inherit properties from rules above them in the hierarchy. Defining groupings of rules allows us to talk about entire sets of rules without specifying the members of the set. This makes writing Tutoring Mode specifications much easier. When we add more rules to a rule class, they automatically are included in the modes which refer to that class. We can also have "meta-rules" which limit the current applicable tutoring rules to one or a small number of groupings, without having to list all of the rules involved (Davis & Buchanan 77). An example of a meta-rule (i.e. a rule about rules) is: "IF the discourse has just started, do not use any rule in the rule-class Topic-Summary-rules."

The inheritance feature makes it easier to write and modify rules, because groups of rules can inherit antecedents or actions from a parent rule class. For instance, if we had a class of rules which encode knowledge about diagnosing misconceptions, the parent rule could have the antecedent "if the goal is to diagnose a misconception", and we would not have to specify that antecedent again for any rules below it in the hierarchy.

A further extension along these lines is representing rules in an object oriented paradigm. Such an implementation would incorporate the grouping and inheritance features listed for frames, and also have procedures called "methods" associated with rule classes (as described in section 4.1.5, and see Stefik & Bobrow 85). Different types of rules may require different types of processing. For example, there may be strategy rules, discourse rules, and

diagnostic rule. Using methods we can specify procedures such as "check for match", "execute action", etc. such that the details of their operations will depend on the class of rules being used.

6.4.3 No "side effects"

In this architecture we restrict the Action specified by tutoring rules to be the name of a Tutoring Action (not a LISP expression to be evaluated). The rule specifies which action to take, but it is not taken right away, rather the action name is determined and the action itself is executed when the Control Sub-system relinquishes control to the Action Sub-system. This allows for a more uniform rule representation and easier tracing of what the system is doing. To further improve tractability and uniformity of control decisions, we will not let tutoring actions call other tutoring actions (they can however, call lower level procedures), and all parameter passing between actions must be done through one of the data bases (which are global data objects). For an action to specify that another action follow it, it must updeate the Control Data Base (for example: put a high priority goal on the goal stack). This specified action may then be executed on the next decision–action cycle.

6.4.4 Tutoring action networks

Woolf (84) and others have noted that human discourse contains certain common patterns. Such patterns of discourse actions (or discourse goals) are hidden when the discourse rules are represented as a large set of canonical production rules. Woolf (84) developed a transition network formalism for representing discourse patterns. This formalism has been recently redesigned in the form of discourse action networks (DACTNs, see McDonald, Brooks, Woolf, Werner 86). Our tutoring system architecture incorporates TACTNs (Tutoring ACTion Networks), which are patterned after DACTNs, but simpler. TACTNs are procedural networks which specify common or default action patterns. See figure 6.2 for an example. They are similar to ATN representation structures (Woods 70), which have been widely used in parsing, but are modified to support planning discourse actions. In ATNs the arcs represent states and the nodes represent tests. In TACTNs the arcs are tests and the nodes are actions. TACTNs combine the antecedent-action formalism of the production rule paradigm used in many expert systems with a recursive procedural network formalism which has been used in planning (Sacerdoti 74).

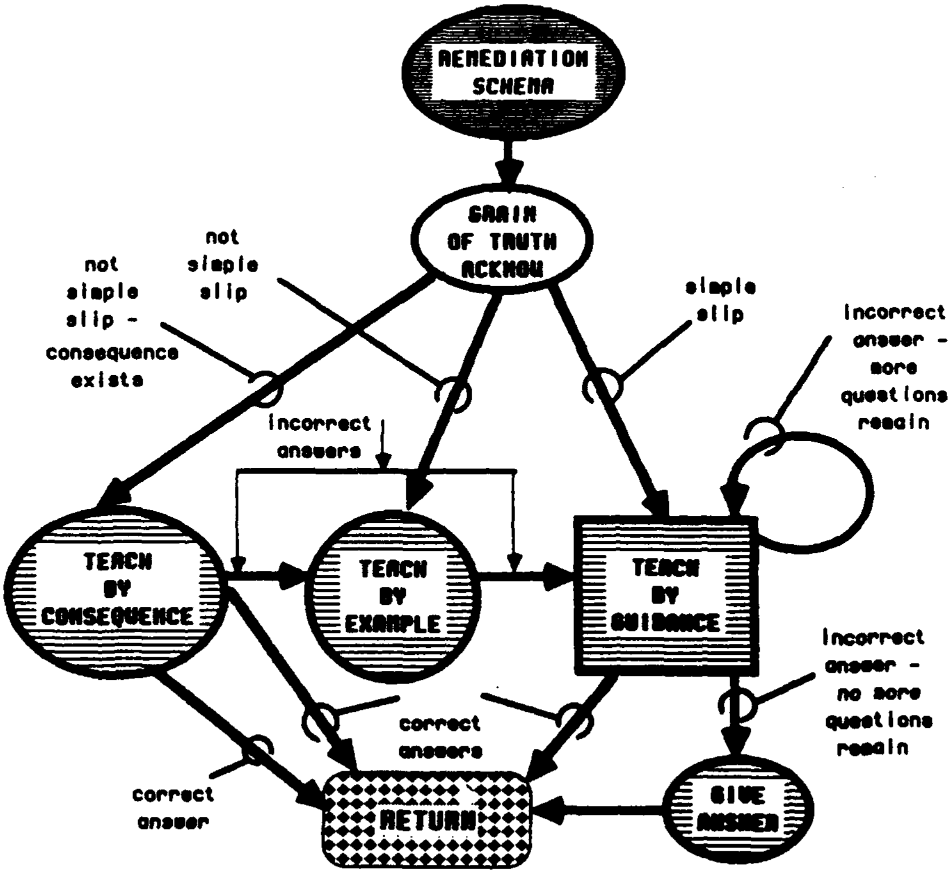


Figure 6.2: A sample Tutoring ACTion Network

The nodes are Actions and the arcs between nodes are complex predicates, called "Situations", which access and test information in the data base. After finishing an action, control passes to one of the children nodes depending on which Situation is true. The links imply rules of the form "IF you just finished doing Action x, and (..the rest of the antecedents...) THEN do Action y". That is, they are like rules which fire in a specific action pattern context. Since the Decision Mechanism need only check for a match between the data bases and few Situations leading from the current Action node, much efficiency is gained (compared with matching with all the rules in a vanilla rule system). Also, the structure of the action pattern is explicit and clear. TACTNs can be graphically displayed for inspection and modification.

In our architecture the Knowledge Sub-system distinguishes tutoring rules from actions, and the Control Sub-system selects one rule and sends it to the Action Sub-system to execute. Therefore, it is not obvious where TACTNs, which combine features of both rules and actions, would fit into the architecture. TACTNs are a subtype of Actions. That is, a TACTN can be specified anywhere an Action can be specified, i.e. by a tutoring rule or as a node of a TACTN. If a TACTN is called, it is "expanded" into its component action pattern.

When a TACTN is invoked (by a tutoring rule or another TACTN), the system continues to cycle through the Decision Sub-system and the Action Sub-system with each Action in the network. When a TACTN is in effect the Decision Mechanism will ignore the search through the entire rule base and choose the Action specified in the network. The reason we continue to cycle through the Decision and Action Sub- systems is that there may be certain control or bookkeeping procedures which need to be done before and after each Action is executed. This also lets us account for the fact that there may be extenuating circumstances where the normal action pattern specified by the TACTN should not be followed.

TACTNs facilitate a three level prioritization of tutoring rules. The arcs coming from the current TACTN node define the default actions in the given context, and are the middle priority level. We can also specify some rules in the rule base as "high priority rules" (or interrupt rules, or daemons, or meta-rules), which get checked before the Situations in the TACTN are checked. For example, we may always want to check that the student hasn't pushed the "quit" button before selecting an appropriate action. If none of the high priority rules fire, and if none of the TACTN arcs fire, then the Decision Mechanism can revert back to looking at the entire rule base for a

match. Thus the rest of the rule base is a third priority level.

TACTNs can have other TACTNs as nodes in their networks. This allows for recursive and subgoal types of control behavior. It is the job of the Control Data Base (below) to keep track of TACTN nesting. The need for subgoal or nested discourse patterns is supported by Grosz (80), whose theory of discourse structure incorporates a hierarchy of "discourse segment purposes."

6.5 The Control Data Base and goal driven tutoring

The Control Data Base stores information relative to the internal control mechanisms of the Control Sub-system. The types of information stored will depend on specific design decisions. Stacks of goals and/or pending tutoring actions will be included. Rather than specify that an action be run, a tutoring rule might specify that a goal or action be put on the stack. Then the Decision Mechanism may re-arrange this stack (sometimes called an agenda) to account for iteractions between goals, or to select the highest priority goal or most urgent action.

We have not said much in this paper about goal driven tutoring, but some representation of tutoring goals may be needed. Gross (86) has shown that some aspects of human discourse can be modeled using goal stacks. This is especially applicable to tutoring discourse. Here is an example of several levels of nested goals of sub-discourses: a goal to "teach Newton's third law" may activate a subgoal of "teaching Newton's law in static situations" which may active a goal to give an example. While giving this example the tutor may diagnose a misconception, and decide to correct it with a counter-example. While giving the counter-example the student may ask for some information. While the tutor answers the student's question, it has all of the above goals still active, waiting for their sub-goals to be completed. I.E. it is simultaneously trying to teach Newton's third law, teach it for static situations, present an example of it, correct a misconception about the example, and answer a student's inquiry.

If the decision making is to be goal centered, then some of the tutoring rules must match against goals. (ex: "IF the goal is to verify a misconception, THEN..."). The Control Data Base can also keep track of the current context, such as the current and recent topics, goals, actions, etc. Part of the complexity of human tutoring lies in the fact that the tutor is simulta-

neously trying to satisfy several goals, such as conveying a concept, keeping the student interested, not offending the student, using time efficiently, etc. It may be possible to have the Decision Mechanism choose an action which satisfies the maximum number of several simultaneously active goals.

6.6 The Decision Mechanism

The Decision Mechanism has the functions of both matcher and planner. The algorithms it uses may be quite complex, and are only suggested in this paper. Matching rule conditions with the data bases must be combined with some conflict resolution algorithm, as discussed above. Since some actions may only add goals or sub-goals to the control data base, pending actions and goals may accumulate. Some of these goals or actions may have higher priority. Also there may be significant interaction amongst them. In such cases it is the Decision Mechanism which must plan actions which most efficiently satisfy the set of pending goals and/or actions.

Part of the job of the Decision Mechanism is to manage the updating and use of the information in the Control Data Base. It uses current goals, TACTN-in-progress flags, etc. in the Control Data Base to coordinate control actions.

6.7 The Action Sub-system

The main functions of the Action Sub-system are to execute the tutoring action chosen by the Decision Sub-system, accept student input, and update the Student and Discourse Models. Figure 6.3 shows the components of the Action Sub-system.

6.7.1 Non-tutoring actions

This component is a catch-all for things such as upkeep of the simulation and maintenance of other non-tutor aspects of the learning environment. We assume that this box is essentially independent of the tutoring control and data, or at least does not change any of the tutor's data bases. In our general tutoring system the simulation and user interface are a components of (or are called by) the tutor.

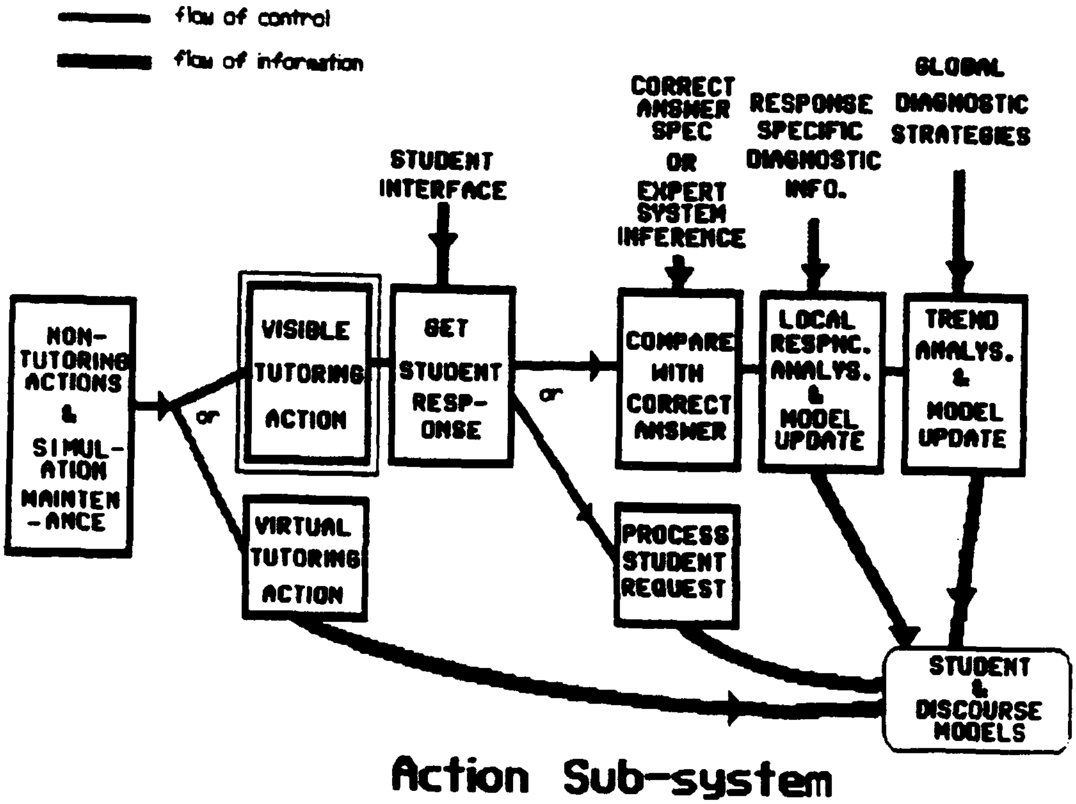


Figure 6.3: The Action Sub-system

6.7.2 Visible and virtual Tutoring Actions

Some tutoring actions are "virtual actions" (or "update actions"). Virtual actions do not make any noticeable change in the student environment. An example would be setting a goal to probe for a misconception. It would be later, when some other action actually did something to satisfy this goal, that the student would see any effect. Visible tutoring actions are actions that do something the student can notice. One exception is the Null action, a tutoring action which does nothing, leaving the student alone and passing control back to the Control Sub-system (hopefully a popular action in most tutoring sessions).

6.7.3 Get student response

After all visible tutoring actions, including the Null action, the computer will check for a student response. The student response component converts the raw information from the simulation or user interface, such as mouse positions, menu choices, typed input, data tables, etc., into some canonical (standardized) form acceptable by the succeeding components. Depending on the type of tutoring action, the Get-student-response component may wait for the student to do anything, or wait for the student to do as much as he wants, until he pushes the "done" button, or not wait at all, and continue on if the student has not done anything since it last checked for a student response. After completion of some tutoring actions control passes to the "compare with correct answer" component (see figure 6.3). This could be as trivial as matching the answer with a string, or a complex inferene concerning the acceptability of the student's response. Get-student-response also looks for student interrupts such as help requests, and processes these accordingly.

6.7.4 Compare with correct answer

This component compares the student behavior with the correct or anticipated behavior and outputs a measure of this comparison. The correct behavior can come from a "correct answer specification" for the problem given, or from asking the expert system what the correct answer is. We recommended designing a standardized language for the inputs and outputs of most of the components in the Action Sub- system. For example, correct answer specs could be phrased in terms of keywords such as "MUST have these three things", or "needs 2 OUT-OF the following 4 things:..., but

MUST-NOT have ..." The output of the compare answer component could be in term of "xx was INCLUDED; yy was OMITTED, and zz EXTRA input was given" (Servi & Woolf 86).

6.7.5 Student and Discourse Model updating

Figure 6.3 shows two components which update the Student and Discourse Models based on the student's behavior. With these two components we distinguish two levels of diagnostic analysis. The "local response analysis" updates the models based upon information in the Task Specs. This information specifies how specific student responses to specific questions contribute to the student model. The "trend analysis" uses diagnostic rules (or, equivallently here, strategies) to analyse patterns of student behavior or belief accross the history of the student's behavior, and accross the entire student model. For example: "if the student's beliefs in concepts X, Y, and Z are under 30 %,.then assert that she has misconception G at 80 %"; or "if the student asks for a hint 5 times in a row, then assert that she is Confused."

6.8 Blackboard archetectures

The architecture presented could be viewed from the perspective of the blackboard architecture paradigm (Nii 86). Blackboard architectures are characterized by having a global data base, opportunistic control, and modularized knowledge sources. They are most useful in situations requiring complex control, opportunistic action execution, and uncertain or conflicting data. In our architecture, the data bases are global, and the various parts of the system share information almost exclusively through the data bases (compared with passing information directly to each other). Control information is contained within a separate complex data structure, as in the Hearsay III blackboard architecture (Lesser 84). The Decision Mechanism prioritizes and selects actions and goals in both bottom up (data driven, as with actions that are triggered by certain patters of student behavior) and top down (goal driven, as with actions triggered as a result of a tutoring goal in the Lesson Specification) inference methods, as in some blackboard systems.

These similarities with blackboard architectures suggest investigating whether our architecture could be configured even closer to the blackboard model. Our architecture is presented in terms of rules and actions, as opposed to knowledge sources. This may be because we have viewed the tu-

toring task in terms of discourse planning, as opposed to problem solving or classification (as would be the case if the overall focus were one of student diagnosis). We will be investigating whether or not the architecture would run more smoothly if represented in terms of modular knowledge sources, each one knowing the conditions under which is was applicable, and the types of conclusions which it could offer.

Chapter 7

TUTORING SYSTEM DESIGN STAGES

Up to this point we have been discussing a general control and knowledge representation architecture for representing and using the various types of information needed in intelligent tutoring systems. Now we will look at the steps that might be taken in implementing this architecture to design a tutor for some specific domain. We will assume for now that an ITS "shell" exists, with the various components as described in previous chapters, and our task is to determine the specific information to be entered into these components. If such a shell does not exist, the steps outlined here can still be used, and the architecture described in previous chapters can be used as a guideline for organizing the information. This chapter and the next deal with knowledge acquisition. This chapter suggests a methodology for discovering and implementing the rules and actions in the system, and chapter 8 suggests a methodology for analysing the domain and the tutoring goals so as to make the ITS more flexible and perspicuous.

7.1 Tutoring rule design stages

The tutoring rules are one of the last components implemented in a tutoring system, because their precise specification requires several other components to be designed first. Figure 7.1 shows a sequence of design activities leading up to the implementation of tutoring rules. In this chapter, for simplicity, we will use the term "tutoring rules" to refer to all kinds and levels of tutoring rules, diagnostric rules, Tutoring ACTion Networks, and tutoring Modes.

Tutoring Rule Design Stages:

- analyze domain / create lesson plan
- generate sample dialogues
- define primitive tutoring actions
- infer vague tutoring rules
- define parameters for domain knowl. bases, student and discourse models
- implement precise tutoring rules
- design diagnostic strategies

Figure 7.1: Tutoring rule design stages

After an analysis of the domain to organize the knowledge to be taught and the general tutoring methods employed, sample dialogues are generated. Sample dialogues could include transcriptions of taped interview studies, dialogues existing in the literature from research on teaching the subject, and/or synthetic dialogues generated by the domain expert/teacher.

Generation of the sample dialgogues is important. The less synthetic they are, the better. Though the effort required to tape and analyse actual or mock tutoring sessions is non-trivial, and certain aspects of the dialogue between the proposed ITS and the student can not be faithfully measured in a non-computer tutoring session.

They provide much design-relvant information which would not be available from a set of un-tried specifications of how the tutor should behave. Also, the process of generating and analysing them helps to reify and shed new light on the initial design conceptions. Sample dialogues should provide information relevant to:

- The variety of types of questions, explanations, or prompts the tutor will need
- The form and content of the inturruptions and questions the student should be able to ask
- The degree of naturalness or mechanicalness of the dialogue
- The degree of interaction (i.e. complexity) between the discourse context and tutoring strategy used

The dialogues (along with annotations by the interviewer) are analyzed in several steps (see figure 7.1). First the primitive tutoring actions are defined. Examples are: congratulate-correct-response, give-extreme-case-example, introduce- new-topic, give-away-correct-answer, etc. Next "vague" tutoring rules are inferred from the dialogue. They are of the form "IF <some condition> THEN <Action>," and are called vague because at this stage they must refer to well-defined tutoring actions (defined in the previous step), but can have anything in the "some condition" part. The vague rules try to explain the conditions which motivated the instances of the tutoring actions in the dialogue. Next we analyze the vague tutoring rules to determine parameters for the Student Model, Discourse Model, and the pedagogical aspects of the Domain Model. For example, if a vague rule says "IF the student is confused THEN give-leading-question," then we need a parameter

in the Student Model called "confusion-state." A vague rule saying "IF the student answered incorrectly too many times in a row, THEN change-focus" requires a "number-of-wrong-answers-in-a- row" parameter in the Discourse Model. A vague rule saying "IF the information being taught is of a factual nature, THEN don't ask probing questions" requires a parameter in the Domain Model for "knowledge-type," which distinguishes factual types of knowledge from other types. A vague rule which says "IF introducing a new topic THEN review-previous-topic" requires a record of the previous topic in the Discourse Model. Next, precise tutoring rules are implemented. Their condition refers to well defined parameters in the knowledge bases, such as "IF student-confused THEN give-leading-question." The actual implementation of the rules may be quite complex, since many of them may be interdependent. Rules may be defined hierarchically, grouped into tutoring Modes, and arranged in networks, as suggested in Chapter 6. The data base parameters may have to be redefined iteratively in order to make distinctions of fine enough grain to enable clarity of the tutoring rules.

The last item in figure 7.1 is designing diagnostic strategies (or rules). Above we have identified parameters in the Student Model such as "is-confused," an "has-adequate-knowledge-of-current- topic." It remains to determine how these parameters will be measured, i.e. what student behaviors will provide evidence for the Student Model parameters (including the expert knowledge "overlay" portion of the Student Model). We do not address diagnostic strategies much in this paper, but their design will typically be one of the more difficult aspects of tutoring system implementation (see Sleeman & Henley 82 and Clancey 86B).

7.2 Tutoring system design phases

In figure 7.2 we elaborate on the activities shown figure 7.1, and include steps in building the knowledge bases, as well as in designing the tutoring rules. This design activities chart is for instruction on the order of one lesson, one topic unit, or one chapter. Figure 7.2 outlines how information from one activity feeds into another. The connections between activities indicate that some of the activities can be going on in parallel. The steps are grouped into 5 phases, roughly outlining these activities: 1. Specification of tutoring goals and strategies, 2. sample dialogue generation and analysis, 3. putting the rules inferred from the dialogue into a computationally precise form, 4. implementing the components of the tutor, and 5. testing and refinement.

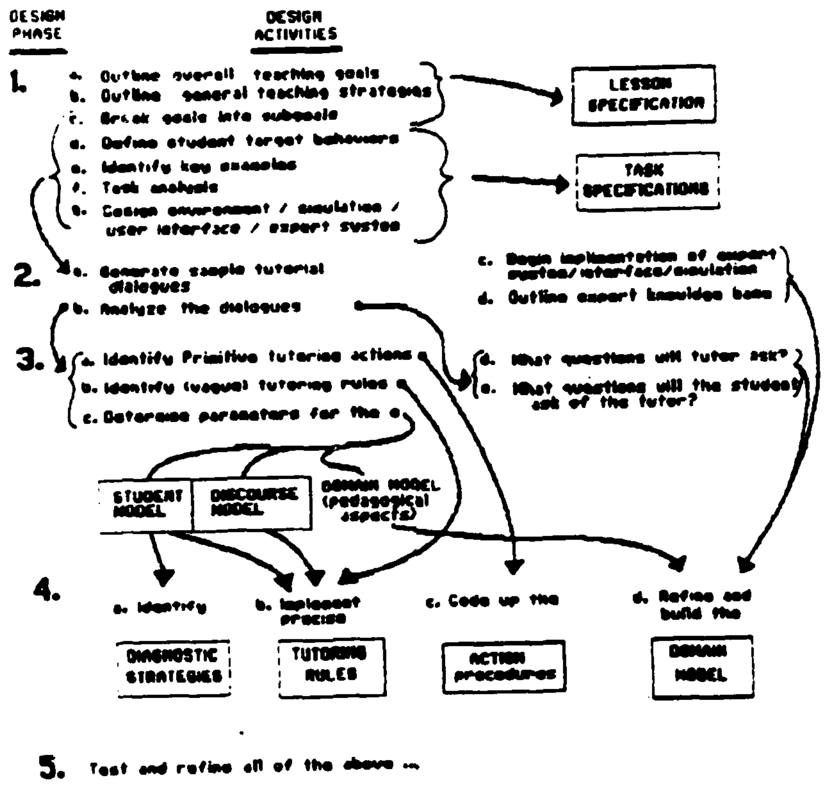


Figure 7.2: Steps in designing an ITS

This activity organization is not intended as a rigid temporal scheduling of personnel tasks. The designers of an ITSs may find that they are skipping all over this chart. The chart gives an indication of what design tasks need to be in progress before others can be seriously attempted.

7.3 Phase 1: Global goal and strategy specification

In phase 1 we first outline the overall teaching goals for the topic unit (item 1a). This includes the important facts, skills, and concepts to be learned. For example, to become familiar with contact forces in static situations, and to be able to solve simple word problems involving static forces. At this stage we also need some general tutoring strategies for our domain (item 1b). For example: using an environment in which the student can create

objects of different weights and sizes and measure the forces they have on each other; the student will be allowed to experiment in this environment, and then the tutor will ask her questions about situations that the tutor presents.

The above are not really planning tasks, they just represent the ideas which motivated the designers to build a computer tutor in the first place. These ideas are next refined. The learning goals are broken into subgoals (item 1c). The goal skills and knowledge are analyzed to produce a network of prerequisite skills and knowledge, as is popular in classroom curriculum design (see Joyce & Weil 72, and Barr et. al.'s BIP tutor discussed in Wenger 86). The more detailed this analysis the better. The student target behaviors are identified (item 1d, discussed in more detail below). Key examples and expected misconceptions are identified (item 1e). The tasks or activities that will allow the learner to build her knowledge and achieve the learning goals are specified and ordered (item 1f).

Also in this first phase, initial specifications for the domain expert system, learning environment (including the user interface), and simulations are drawn up (item 1g).

7.4 Phase 2: Sample dialogue generation and analysis

The simulation and/or learning environment is designed so that coding of these can begin (item 2c). "Knowledge acquisition" of the domain knowledge starts, i.e. putting the knowledge in terms appropriate for a computer knowledge base. The knowledge hierarchy developed above is refined and organized into knowledge units. Types of links between the knowledge units are specified (item 2d).

Occurring simultaneously with the above, the knowledge organization and learning tasks defined in phase 1 are used to draw up a primitive lesson plan which can be tried on students. Sample tutorial dialogues are obtained as described above (item 2a). These dialogues (and/or the interviewer's comments of the tutorial sessions) are analyzed for the purpose of obtaining the information in phase 3 (item 2b).

Once the simulation/learning environment is implemented, it should be used to generate more sample tutorial dialogues. At this phase the computer is used as a tool for simulating the domain, presenting examples, experimenting, etc., and the interviewer does all tutoring and analysis of student actions

(i.e. he takes the place of the Decision Sub-system described in Chapter 6).

7.5 Phase 3: Defining the tutoring rules

From the dialogues we would like to infer the tutoring rules which are being used. Referring to figure 4.3 (the Control Sub-system), we can describe the tutoring rules as a mapping from data about the domain (the static information) and about the current context (the dynamic information) to a set of potential tutoring actions. (The Decision Sub-system figure 4 carries out the mapping which the tutoring rules define.) That is, there is a correspondence between all possible tutoring situations, and all possible tutoring actions, and this correspondence is specified by the tutoring rules, which (we assume for simplicity) are of the form "IF <condition> THEN <action>." There are three things to define: the set of actions, the set of conditions, and the set of rules (which are a mapping from the conditions to the actions).

In phase 3 we analyze the dialogues to define primitive tutoring actions and the parameters for the Student Model, the Domain Model, and the pedagogical aspects of the Domain Model, as described above and shown in figure 9 (items 3a, 3b, and 3c in figure 9).

Also in phase 3, somewhat independent of the above, is an analysis of the discourse to determine what kinds of knowledge and questions the tutor should be able to ask, and what kinds of questions about the domain knowledge the tutor should be able to answer. This information is needed to better define what knowledge is needed in the domain knowledge base, and how it should be represented.

7.6 Phase 4: Implementing the system components

Phase 4 represents the point at which the code for the various components is implemented into computer knowledge bases or LISP code. (The Student and Discourse Models are boxed in phase 3 rather than phase 4 because they do not contain knowledge about the student or discourse until tutoring actually begins. Only their structure is defined.)

After a precise vocabulary for describing tutoring actions and parameters is complete from phase 3, the Tutoring Rules can be implemented. As mentioned above, this will include defining Tutoring Modes and Tutoring ACTion Networks. Making tutoring rules precise may take many iterations.

For example, the vague rule "IF the student is confused THEN give a hint," may eventually become "IF the student is confused on the current KU while the tutor is presenting new knowledge, and she hasn't been given any hints before, THEN give a level-one hint." We must evaluate the many possible interactions and the possibilities of rules appearing to be applicable in situations where it should not be.

The Domain Model is implemented according to the analysis of the expert knowledge and the sample dialogues, as indicated in figure 9. Diagnostic strategies can then be incorporated into the Task Specs and the Domain Model (see section 4.2).

Phase 5 makes explicit the iterative nature of the design process, and includes the obvious activity of analyzing what we have so far, and if we are satisfied, trying it out on some students and repeating the whole process to refine the system.

Again, figure 10 does not suggest a step by step sequence of activities, but shows conceptually how the design of different components are related. At every step along the way the designers will probably be noting ideas about activities lower in the diagram, and be circling back to refine earlier activities. The diagram also highlights the importance of the sample tutorial dialogues, form which much of the information is gleaned.

Chapter 8

KNOWLEDGE ENGINEERING FACTORS

In the previous chapter we discussed how tutoring rules could be designed based on an analysis of sample tutorial dialogues, which were in turn planned according to the teaching goals of the topic to be tutored. We implicitly assumed that the tutoring rules implemented in a computer tutor attempt to duplicate the behavior of human teachers. However, many of the tutoring rules and other design decisions incorporated into tutoring systems will be based on principles and assumptions about teaching in the domain, not on an analysis of (real or synthesized) human tutorial dialogue. Even in the case of tutoring rules which are patterned after discourse protocols, there are assumptions about teaching and learning in the domain behind the lesson plan and the dynamic decisions made by the human tutor. In this chapter we discuss ways of evaluaating the many factors which come into play in ITS design, some of which are usually implicit. We suggest several perspectives for the analysis of design factors, and give examples of how this information might effect the design of the system. We also advocate an explicit accounting of the design decisions made by research teams.

In looking at the many tutoring systems in the literature, it is apparent that no two of these incorporate the same rules about effective tutoring. Few tutoring systems have both an explicit philosophy of tutoring and learning and an explicit realization of this philosophy into tutoring rules (but see Anderson's research). For the sake of system clarity, modification, and evaluation (as discussed in chapter 2), it is desirable to be explicit about both the overall philosophy behind a tutor, the rules used to guide the tutorial

discourse, and the assumptions behind each of the rules. Also, it is only by being aware of the underlying assumptions behind the rules of a given tutoring system that we can decide whether we can incorporate similar rules into a new system.

8.1 Rules, principles and assumptions

Appendix 1 shows tutoring rules and principles from some well known tutoring systems. One observation is that they don't all agree on how to tutor. This is not surprising, considering the variety of domains being tutored, and the variety of underlying psychological, pedagogical, and philosophical assumptions (both explicit and implicit) supporting the design of each. Another observation about these rules and principles is that they describe factors on many different levels. Some rules involve psychological or philosophical assumptions; some rules are given without explaining why they should work; some refer to vague strategies, and others to specific actions. Some are more relevant to a particular domain than others. As an illustration of a continuum of specificity of tutoring rules or principles, consider the following ("English-ized") rules from hypothetical tutoring programs:

1. *If question 12 is answered wrong, give explanation 5.*
   This is the type of very specific coupling of diagnosis and action found in (non-intelligent) computer assisted instruction programs.

2. *If the student gets more than three questions wrong in a row, then do a remedial exercise for the current topic.*
   This type of rule is more typical of intelligent tutors. The system must monitor the student's answers and infer which remedial exercise to give. Note that it says nothing about the reasons why the rule is applicable.

3. *If the student is very CONFUSED, then GIVE-HINTS for the current topic.*
   (Assume here that the property CONFUSED is determined by some low level data, such as the number of wrong answers, and GIVE-HINTS is interpreted differently for different topics.)
   This rule is at a more abstract level. Abstraction makes the rules more transparent, and allows a separation of the rules themselves from the details of how the conditions and actions are implemented. A diagnostic strategy must infer the level of confusion from student behavior

(such as number of times asking for help). The GIVE-HINTS action is a general action which gets interpreted differently for different types of topics.

4. *If a student answers a problem incorrectly, give a series of hints, from vague, to more specific, to giving away the answer. Give the student several opportunities to think again about the situation, and learn from mistakes, before giving authoritarian feedback, which the student can accept and memorize without critical thought.*
   This represents the principle behind the previous rule. It states a pedagogical bias or strategy, and is not precise enough to be part of a computer program (with today's technology).

5. *People learn through an active process of concept formation while trying to account for new observations in the context of their previous knowledge.*
   This is a theory—a psychological, or philosophical assumption. It represents the reason for the previous principle and the purpose for the rule above it.

Note that one can (barely) conceive of a computer tutor which could take principles and infer tutoring rules, or even take psychological assumptions and infer principles and rules. This of course is stretching it, but points out that as we progress down the list, more intelligence is needed to translate the rule into a specific tutoring action. We suggest that ITS rules be implemented on the level of item 3, and that all such rules (and other design decisions involved in system design) are explicitly annotated with the principles and/or assumptions which are behind them (as in items 3 or 4). This allows systems to be evaluated and modified on the bases of their underlying assumptions, and allows tutoring systems to explain why they are performing certain tutoring or discourse actions as they do so. The ability to explain tutorial discourse is beneficial for testing the system and for exhibiting computer "candidness" to the student.

8.2 Introduction to an analysis of the factors affecting tutoring system design

In designing new tutoring systems, we would like to glean the most effective tutoring rules from the research of our colleagues who have designed and

tested these systems. Our question is not "what are the correct rules and principles for tutoring?," since these will differ according to domain and tutoring goals. Our question is not even "what are the correct rules for my domain?," since, as discussed in Chapter 5, a variety of tutoring styles (or Modes) should be incorporated. What we want to know is "which rules are appropriate in each particular tutoring situation?" In order to answer this question by looking at existing systems, and in order to organize design decisions concerning appropriate tutoring rules for new systems, we will present an analysis of the factors involved in describing tutoring situations and assumptions.

The left side of figure 8.1 shows four types of factors involved in the design of tutoring systems. These factors are the (implicit or explicit) assumptions about the domain, the learning goals, and pedagogy, which underlie the specific contents of the various components of the system. The behavior of the tutoring system is determined by the contents of these system components (as shown in the right side of figure 8.1): the tutoring Rules (including meta- rules, TACTNs, diagnostic rules, etc.), the Lesson Specification (which specifies the overall teaching goals, lesson plan, and tutoring modes), the examples and student tasks specified in the Task Specifications, and the simulation / leaning environment / user interface. In *all tutoring* systems, regardless of their architecture and components (i.e. whether of not they have the components shown to the right in the figure), assumptions in the four areas show to the left of figure 8.1 underlie the design and behavior of the system. We will discuss each of the four areas below. The categories given are somewhat fuzzy and overlapping, but we present them as distinct categories to provide a framework within which to do the analysis.

The "**pedagogical characteristics of the target knowledge**" are determined through an analysis of the differing needs and constraints for learning different types of knowledge. Many of these characteristics and the reasons why they are useful, have been discussed in the Chapter 4 (on knowledge representation). For example, fact/skill/concept distinctions, and distinguishing between qualitative and quantitative knowledge. In this chapter we will outline some finer distinctions based on pedagogical needs.

All domains seem to require a wide variety of types of target knowledge, such as facts and skills, so rules referring to types of target knowledge (mentioned above) are fairly general and applicable to many domains. These rules determine the dynamic decisions made by the tutor. Domains also have predominating characteristics which determine the overall design decisions for tutors, such as what general tutoring strategies will be used, how the simu-

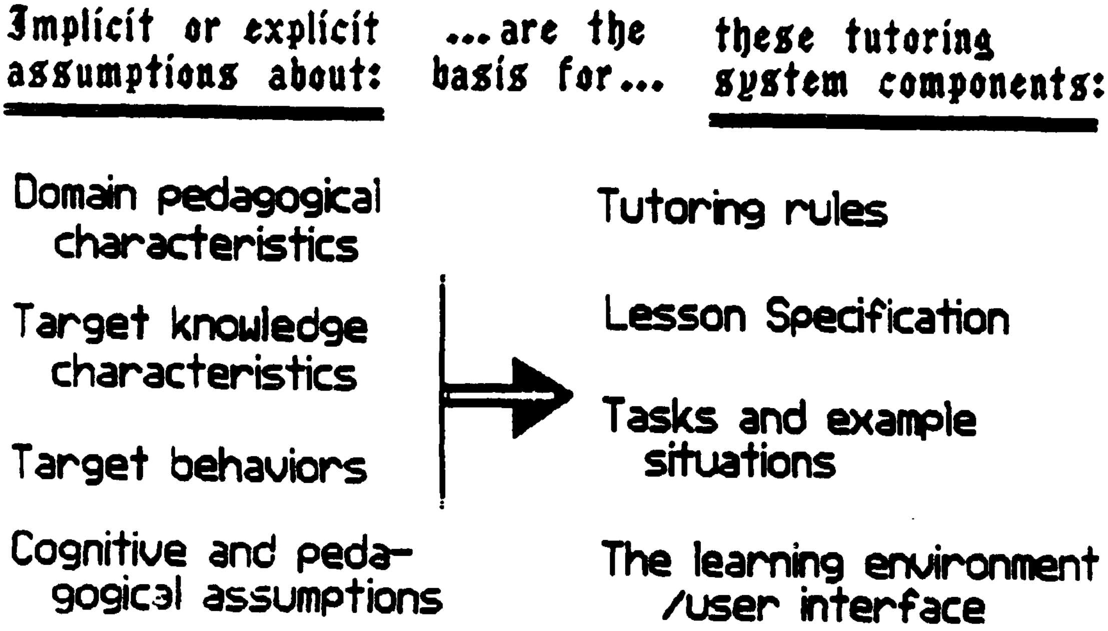


Figure 8.1: Factors involved in the design of ITSs

lation/learning environment/user interface is designed, and the contents of the Lesson Specification.

The **"pedagogical characteristics of the domain"** refer to characteristics which determine the overall approach for a tutor. For example, even though tutors for electronic circuit design and LISP programming may have similar rules about how to teach certain categories of knowledge, there are probably significant differences in their most auspicious overall tutoring approaches. To make clearer the distinciton between domain characteristics and target knowledge characteristics: both domains use many types of knowledge. Some types of knowldge, such as "memorized algorithms," may be common to both domains. Even though considerations about domain idiosyncratics determins the design of some tutoring strategies (such as how much time should be spent in coaching environments), there may be other strategies (such as how to teach memorized alogirthms) which depend only on the type of knowldge, and are independent of the domain.

The **"target behavior"** item in figure 8.1 refers to a specification of what student behaviors are expected as the result of successful learning. It is not enough to specify only the target knowledge–"the laws of electricity," for example. This alone leaves an uncertainty concerning how we will know when the student knows it, and to what depth we expect to teach it. Using specific target behaviors (such as successfully solving problems, or answering questions) aids in evaluating the effectiveness of the system, designing diagnostic rules, and focussing the design of the system so as to encourage these behaviors in the student. Note that tutoring system design can be guided by either target behaviors or target knowledge (or both). Tasks (which specify behavioral tests in their questions for the student) can be incorporated into the system with the goal of teaching and testing certain target knowledge units, or, alternatively, the knowledge units incorporated can be determined based on what knowledge the student will need to exhibit a certain target behavior.

The last item in figure 8.1, **"cognitive and pedagogical assumptions,"** refers to the biases and assumptions about learning and teaching which the designers hold. For the most part these are taken as true, and are difficult to verify experimentally to the satisfaction of all. These assumptions are the ideological foundations upon which the design of a system is based. We do not advocate any particular philosophy, but suggest that the assumptions be made explicit, and present some examples.

The initial design phase of a tutoring system should include consideration of factors from all four of these area. Below we will enumerate a few

factors from each area, and show how they affect the design of some common tutoring systems.

8.3 Pedagogical characteristics of the domain

Teaching about physical systems

Domains which involve physical systems, man made ones (such as circuits and steam boilers–SOPHIE and STEAMER) and naturally occurring ones (meteorology and medicine, WHY and GUIDON), may require that the tutoring system be able to generate sophisticated explanation about causal and functional relationships (as in WHY). Man made systems will be more focussed on the function or purpose of the components. Natural systems will focus more on the causal relationships between components. To contrast, teaching abstract (man-made) systems, such as programming (PROUST and The LISP Tutor), geographical facts (SCHOLAR), and language (CALLE (Xerox 85) and CALEB (Cunningham et. al. 85)) may not need complex explanation facilities.

Domains prone to misconceptions

Misconceptions can occur as a result of non-academic experiences, or during instruction. Misconceptions which arise during instruction can result if a concept or skill is over-generalized, over-specified, or just mis-understood. White & Frederiksen (86) argue that these types of misconceptions can be greatly reduced with careful sequencing in instruction. However, learning in some domains, such as physics (Clement 85, McDermott 84), probability/statistics (Tversky & Kahneman 74), and programming languages (Soloway & Johnson 84, Bonar & Soloway 85), can be severely impeded by the influence of beliefs or patterns of inference developed quite independent of any academic context (see Claxton 86 and diSessa 85). In such domains tutoring rules must incorporate detection and remediation of common anti-productive beliefs.

Problem solving domains

In some domains (or parts of domains) the focus is predominantly on problem solving, rather than learning new facts, skills or concepts. Examples of tutoring systems in these domains are SOPHIE, GUIDON,

The Geometry Tutor, and Keynesville (Shute & Bonar 86). These tutors pay explicit attention to the problem solving process (diagnosis, proof, or rule induction), as well as the specific rules and heuristics used to solve the problems. Student modelling (if done) is relatively more illusive in these domains.

Information density

Domains such as programming (in The LISP Tutor) and medical diagnosis (in GUIDON) can be characterized as having many rules, each with relatively straightforward application (here "rules" refers to the human problem solving rules, such as those found in a textbook, not computer program rules). Domains such as physics and electricity have relatively fewer rules or concepts. In these domains the rules can be easily learned, but applying them properly can be less straightforward. In domains of low information density, there is more need for examples showing multiple viewpoints (as in WHY), non-examples, and boarder cases (Rissland, Valcarce, & Ashley 84, and Tennyson & Park 80).

8.4 Pedagogical characteristics of the target knowledge

Newness of the knowledge

If the subject being taught is new to the student, tutoring rules may be needed which introduce it, piece the new knowledge together in a systematic way (SCHOLAR, LISP, CALLE, CALEB), or allow the knowledge to be discovered (Keynesville, WEST, WUMPUS, MENDEL). If topic is not new to the learner, the tutoring may focus more on diagnosing misconceptions and

giving the student a chance to practice using the knowledge in a variety of contexts (as in SOPHIE, GUIDON).

Memory intensive vs. inference intensive knowledge

When asking questions about information which requires primarily recall or recognition of facts or patterns, the strategy of immediate feedback may be best (Anderson 84, and *The Little Lisper*, a programmed learning text). When asking questions which require more deductive or creative thought, it may be advisable to let the learner

learn from mistakes, giving only enough information to let the learner debug his own knowledge.

Other characteristics: Meta- and qualitative knowledge

In chapter 4 we gave reasons for distinguishing between qualitative and quantitative knowledge, and between knowledge and meta-knowledge. Appendix 2 has a detailed breakdown of types of knowledge for pedagogical purposes (compared with Chapter 4, which categorized knowledge according to knowledge representation needs). It is included to suggest a taxonomy which may help distinguish different types of knowledge. We will not discuss it in detail however, since at this point there is no clear science for tutoring methods based on different knowledge types of this fine a breakdown.

8.5 The target behavior

Solving unfamiliar problems

Teaching such that the student can solve routine problems in a domain differs greatly from expecting her to solve unique problems, i.e. problems that don't fit nicely into any previously learned pattern (Chi et. al. 81, Larkin 83). The level of difficulty and/or depth of learning which is expected for any given knowledge unit is best specified by listing examples of the types of problems which the student is expected to solve. Some tutoring systems, such as WUMPUS, Whites electricity tutor, and WEST, pay specific attention to the various levels of expertise in understanding a knowledge unit.

Target behavior for memory intensive learning

In specifying the target behavior for knowledge which is predominantly memorized, such as facts, simple skills, formulas, relationships, etc., a distinction can be made between expecting (in order of difficulty:) recognition ("Is this a marsupial?"), associative recall (give a hint or strongly associated context: "are all things with hair marsupials?"), free recall ("tell me all the kinds of marsupials"), and generation (creating a new exemplar of the knowledge type). Which of these is expected will effect how the knowledge is taught (Cofer 79). For example, knowledge always presented in a limited context may not be retrievable in a different context.

Target behavior for domain-independent skills

Tutors which attempt to teach domain independent complex skills, such as investigative/inquiry/experimental skills (Keynesville, SOPHIE), or logical inference skills (such as WUMPUS) may have to give students tasks from several domains before they are sure that the student has such a skill. This is not currently done.

8.6 Cognitive and pedagogical assumptions

Using examples

Tutoring systems differ appreciably in the importance they place on examples and how they are presented. Rissland (83) stresses the importance of presenting examples as well as definitions of concepts, and Tennyson & Park (80) give psychological data implicating their importance in the learning process. In WEST, providing examples for the "issues" taught is key. Collins (77) and Tennyson & Park have techniques for selecting and ordering examples, and using counter-examples and examples which focus on specific features.

Teaching mental models

Several research groups stress the importance of assisting the student in the construction of runable mental models of the system under study (deKleer & Brown 82, White & Frederiksen 86, Papert 80). Tutoring under this assumption involves asking the student to make predictions about the behavior of the system under different conditions, and allowing the student to play with a simulation of the system in order to intuit the patterns of its behavior and a causal model of its operation.

The role of feedback

Opinions differ concerning the type and frequency of feedback needed in tutoring. Anderson, as well as those in the behaviorist tradition, emphasis the importance of immediate feedback. Others, of a more constructivist bent (as mentioned above) would rather err on the side of giving too little feedback, since too much can effect the student's motivation and cognitive engagement. Most researchers have some allocation for giving hints before giving away an answer, but the degree differs. Burton & Brown (82) and others have mentioned the importance of positive as well as negative feedback.

Constructivist paradigms

There are many consequences resulting from a constructivist (explained

in a previous chapter) theory of learning. Constructivism is a "paradigm" (as is behaviorism), not a tight theory, and tutoring strategies based on it can be quite diverse and even incongruent. Some (Clement, McDermott, and others), focus on the importance of recognizing and remediating "deep" misconceptions, persistent erroneous intuitions inferred from common experiences. Some focus on the need for accessible learning "tools" and rich environments, giving the learner maximal control over the learning situation. Some (White and Frederiksen and others) stress the need for thorough analysis of prerequisite knowledge, and careful sequencing of the topics taught. Some stress the need to encourage in the learner a state of cognitive dissonance (or disequilibrium) (Lawson and others), which will motivate the learning process. Some (Polya and others) stress the need to apply new knowledge in realistic problem solving context. Some (such as Papert and Clement) mention the importance of anthropomorphizing the concepts; i.e. encouraging the learner to put himself in the place of the gear or the ice puck in the problem.

Theories of the mind

Anderson seems to be the only ITS designer to date who is basing his tutors on a specific cognitive theory. The theory includes assumptions about the limitations of working memory (which result in design decision to give the student a larger effective working memory), and assumptions about the structure of knowledge in the mind. Part of Anderson's theory (the ACT* theory of cognition (Anderson 83)) is that human skills can be modeled using a set of production rules (if-then rules), and this forms the basis for many design decisions (Anderson, Boyle, Farrell, & Reiser 84). (Others have represented skill knowledge as production rules in tutoring systems, but have not designed entire tutors based on a cognitive science theory, as does Anderson.)

Chapter 9

CONCLUSIONS

9.1 Review

We have given many suggestions aimed at systematizing the design of Intelligent Tutoring Systems, and providing some common ground on which to compare systems and share research ideas. After outlining some problem areas in the field, and the research goals which these suggest (Chapter 2), we presented architectures for knowledge representation (Chapter 4 and 5) and program control (Chapter 6). We then outlined how these architectures could be used in implementing a tutor in some domain (Chapters 7 and 8). Along the way we have introduced terminologies and classifications to help make the process more precise. At UMASS we have begun to implement these ideas.

9.2 Contributions

This paper has both summarized past work in the field and offered new suggestions. As such it serves as an introduction to the literature, an introduction to the main concepts in ITS design, an analysis of previous work in them, and a recommendation for improvements. The ideas herein come from the literature and our experience at UMASS implementing tutors and general components of tutors. Therefore some of the suggestions are culled from the literature and others are original. Below we will highlight the portions of the paper which are innovative or potential new contributions (these claims are accurate only to the extent of my familiarity with the ITS literature). The claims that certain ideas are original pertains to within the ITS

research literature only. I assume that concerns similar to ours are spawning similar ideas elsewhere. Also, we pick and choose from several emerging AI technologies. This paper does not contain contributions to the fields of artificial intelligence or educational psychology, but we apply research in these fields to the field of intelligent tutoring systems in perhaps novel ways.

- The control architecture proposed in Chapter 6 (figures 6.1 and 6.3) is original (although the individual AI methodologies, such as object-oriented programming, meta-rules, etc. are all well established technologies.)
- The Tutoring Action Discourse Networks, as a means for describing common or default discourse patterns is original (Woolf & Murray 86).
- The "High Level Lesson Planning Specification" Chapter (5) is predominantly original. There has been little research effort going into designing ITS systems flexible enough to teach in a variety of diverse "modes" or tutoring styles, teaching different aspects of a unit of knowledge in different ways, or at different levels. (Although the fact that there is a spectrum of tutoring styles in ITS's has been often discussed, for example, in Wenger 86).
- The introduction of "Tasks" (containing example situations and task associated with them) as entities in a separate knowledge base (separate from actions and expert knowledge) is new. The introduction of an explicit "Lesson Specification" data structure may be new, but I think that very similar constructs exist implicitly in some ITS systems.
- The Action Mechanism (figure 6.3), embedding the execution of each tutoring action which the tutoring rules select within a structure which contains ubiquitous action procedures (such as get-student-response), may be new.
- The division of diagnostic components into local response analysis and trend analysis (figure 6.3) is new.
- The categorization of knowledge distinguishing deep concepts and nexus concepts is new.
- A general methodology for design and knowledge engineering for arbitrary tutoring systems, such as the one given in Chapters 7 and 8, has not been seen in the literature, though many ITS system designers may well have similar ideas.

- The focus on a general system architecture or authoring shell is not unique (Bonar is working on it, and Clancey has suggested it), but it is quite uncommon.
- The concern for a more precise terminology with which to describe types of knowledge and actions is also uncommon (but see Clancey 86B). Our way of taxonomizing knowledge and the factors effecting knowledge engineering may be useful.

9.3 Remarks on ITS evaluation and the effects of ITS novelty

The art of designing Intelligent Tutoring systems is in its infancy. Though the field has existed since approximately 1970 (Carbonell 70), reports of systems being used successfully to teach non-trivial samples of students are only recently being rumored (but see Suppes' EXCHECK tutor described in Wenger 86). This is understandable. It took many years for those using new technologies such as the book or the television to agree on felicity guidelines. The computer as a learning medium will have a similar, though perhaps accelerated, learning curve. The design of successful ITS systems taxes the state of the art in AI technologies such as knowledge representation, interface design, machine learning, and natural language processing. Also, codifying the principles of good human tutoring will be difficult, considering that even for classroom or textbook teaching, it is hard to find research evidence for teaching principles which have proven to be successful. However, the existence of computer tutors will contribute to the analysis of teaching and learning in general, since using computers eliminates many uncontrollable factors involved in human-human interactions, and because attempts to codify tutoring rules forces instructional experts to examine what they are doing at a new level of detail and clarity.

Because of the newness of the field, we should be cautious about evaluating systems which present the student with many options. Learners (school-aged and adult) are not used to being immersed in such flexible instructional environments. Students will now be able to manipulate these novel learning environments in ways impossible with textbooks or lectures. Here are some examples of the power learners can have over their computer tutoring systems (given in informal natural language, but they could also be menu items):

- Effecting the rate of delivery:
  "Please slow down, I'm confused" (or "speed up, I'm bored").
  "Please give a more detailed explanation of that example."
  "Stop here–I'll be back tomorrow."

- Exploring the space of knowledge:
  "How does entropy relate to the stuff in the last chapter?"
  "What concepts do I need to have before I can understand entropy?"
  "Can I preview what's coming in the next chapter?"
  "What does 'vectored interrupt' mean?"
  "List all the formulas involving friction."

- Asking "meta-dialogue" or pedagogical questions:
  "Why did you give me that example?"
  "Is this topic difficult to learn?"
  "What facts do you think I don't know at this point?"

- Choosing the material presented:
  "I want to try the next experiment now."
  "Please give me another example of convex polyhedra."
  "Give me a trace of how you concluded that disease from the data."

- Choosing the teaching mode used:
  "I learn better if you give quick feedback for wrong answers."
  "Could you hold my hand through this one–explaining each step?"
  "Don't give me hints so fast, I want to think it through myself."

- Experimenting and "playing" in rich, complex, or realistic environments or simulations:
  "What if we replaced the law of gravity with a different one?"
  "Now I'm approaching the third moon of Jupiter–fire left thruster number 2 for 3 seconds."

But having all this power available does not mean it will be utilized. Learners are, for the most part, not used to this kind of control. They are not used to monitoring their learning or problem solving progress (Confrey 85) and/or altering their learning environments according to reflective metacognitive analysis. When we evaluate the success of these powerful environments using random students, we are sure to find that they under-utilize the system, and do not learn as much or as quickly as expected. We may need to first instruct students in how to make full use of the potential of such

systems; how to explore, play, question, and summarize; how to monitor their own progress; how to communicate with a computer tutor on a "meta-dialogue" level. Fair evaluations of ITS's may only come with a generation of students who have used them several times.

On a similar note, we who are in the business of designing intelligent tutoring systems are under somewhat of a handicap. We (99.9 percent of us) were not taught using ITS's. Our education experiences have been traditional, despite our belief in the power of alternate leaning methods. As we design our systems we can only hypothesize about the experiences of a novice in some domain sitting at a computer terminal and interacting with it. Compare this with textbooks and lectures. We all have our own guide-lines concerning how to write papers or give talks so that the audience will be attentive and motivated, and learn most effectively. We have these guide-lines because we have been reading books and listening to lectures for years, making mental notes on what is effective and what doesn't work. The vast majority of us have not even once learned a new topic in some field via a computer (not to mention an ITS). Perhaps the next generation of ITS designers, based on their experience using the systems we are now building, will produce intelligent tutoring systems beyond our imagination. Of course, if they build systems that match what we've imagined, that would be quite a feat itself.

Appendix A

SAMPLE TUTORING RULES AND PRINCIPLES

Below we show tutroing rules and principles from six well known ITS research projexts. (Note: Reasons for tutoring rules marked with a "*" are reasons which I inferred based on a description of the system.)

1. Stevens & Collins (77,82, Collins 77); SCHOLAR and WHY. No reasons are given for these. These rules were gleaned from analysis of Socratic tutoring dialogues. The main pedagogical assumption is that the Socratic technique is effective for teaching specific exemplars, causal dependencies, and reasoning skills.

   (a) If the start of a dialogue then ask about a known case.

   (b) If the student gives an explanation for a factor on a causal chain where there are also prior factors, then ask for the prior factors.

   (c) If the student gives as an explanation one or more factors that are not necessary, then present a counter-example.

   (d) If the student is missing a particular factor, then show an extreme case of this factor.

2. Burton & Brown (82); the WEST tutor:

   (a) Do not tutor until the student has a chance to discover for himself as much of the structure of a situation as possible. Reason*: Constructivist paradigm–knowledge is constructed from experiences according to existing knowledge; also, be conservative so as

not to interrupt the student too often, as in when he is making simple slip rather than exhibiting a fundamental misconception or lack of knowledge.

(b) Provide concrete examples of, as well as descriptions of, concepts (or "issues").

(c) Before giving advice, be sure the issue is one in which student is weak. Reason*: Inappropriate advise or criticism inhibits motivation.

(d) After giving advice, allow the student to try to use that advice immediately. Reason: To encode the new information most effectively in "episodic memory."

(e) If a student asks for help, provide several level of hints. Reason: Forces the student to continue to be mentally engaged and gives repeated opportunities for him to discover.

3. Goldstein (82); the WUMPUS tutor:

   (a) Assumption: Knowledge evolves along genetic links–from specification to elaboration, deviation to correction, abstraction to refinement, and specialization to generalization.

   (b) Give highest priority to teaching skills at the "frontier" of knowledge. Reason*: These are not to difficult or too trivial. "...learning is facilitated by being able to explain a new skill in terms of those already acquired (pg. 64)."

   (c) Give multiple explanations for a topic, corresponding to the different links attached to it.

4. Clancey (82) GUIDON:

   Clancey describes three types of tutoring rules: 1. for selecting discourse patterns, 2. for choosing domain knowledge, and 3. for maintaining communication module. The rules shown in the paper are quite specific to the domain, and we refer the reader to the article.

5. Brown, Burton & deKleer (82); SOPHIE:

   (a) Teach in the context of problem solving using a simulation. Reason: Experiential learning "capitalizes on episodic memory" so

that the experience is "anchored to a personally meaningful context." "The problem solving process gives rise to experience that structure the factual knowledge" (pg. 228).

(b) First watch the expert solve a problem. Reason: Gives the student a "graceful introduction" to the system being studied, and the types of reasoning involved.

(c) Let the student learn form mistakes; Encourage him to "formulate, test, and witness the consequences of his ideas." Reason: "Every time the student makes a wrong prediction, he has an opportunity to go through the "What? That can't be! Aha¡' cycle which improves the accuracy of his world view" (pg. 233).

(d) If the student produces an effect that illustrates a principle that he has just learned, and does not notice the connection, the system should point it out and give opportunity to change directions.

(e) Expose student to alternative ways to solve a problem. Reason: This more correctly demonstrates how an expert solves problems, considering several methods before choosing one.

6. Anderson (Anderson, Boyle, Farrell & Reiser 84); the *LISP Tutor* and the Geometry Tutor :

   (a) Tutor according to the goal structure of problem space. Reason: "Problem solving behavior is organized around a hierarchical representation of goals" in the mind.

   (b) Provide instruction in the problem solving context. Reason: "Memories are associated with the features of the context in which they are learned."

   (c) Provide immediate feedback for errors. Reason: To make the learning process more efficient; when feedback is delayed, it is harder for the student to properly assign blame to the faulty behavior.

   (d) Provide tools which increase the student's effective working memory capacity. Reason: some errors in problem solving are due to working memory overload–minimizing these errors allows the student to focus more efficiently on the domain knowledge.

Appendix B

A KNOWLEDGE TAXONOMY

(A somewhat ad-hoc taxonomy of types of knowledge, showing how such a taxonomy might be structured. Categories actually implemented in a tutoring system should distinguish features of the knowledge that are relevant to tutoring rule decisions.)

1. Primitive knowledge types:

   - needed for complex skills
   - little or no uncertainty
   - student demonstration of knowledge or ignorance is straightforward
   - can be demonstrated out of context–without being 'used'

   (a) facts
      - values (memorized numbers, dates, names, etc.)
      - formulas, laws, rules
      - definitions

   (b) relationships (2-place predicates)

   (c) algorithms, procedures, simple skills

2. Complex knowledge:

   Schemas of primitive knowledge

(a) deep concepts (gestalts)
(b) nexus concepts (structural combination of other knowledge)
(c) heuristic knowledge (fuzzy skills)
(d) runable mental models (with parameters)
(e) ubiquitous non-domain-dependent knowledge (concepts and skills): logic, probability, causality/correlation, estimation, inference skills, conservation.

3. Meta-knowledge

   Ancillary knowledge about the primitive or complex knowledge units

   (a) source of a knowledge unit (defined, empirical, practical)
   (b) limitations and assumptions beneath a knowledge unit
   (c) intended uses, purposes of a knowledge unit
   (d) role within a larger context
   (e) knowledge about learning, refining, verifying the knowledge unit (how to, when to)
   (f) personal limitations related to the knowledge unit - working memory limitations, uncertainty of correctness, weak pre-requisites
   (g) coordination of multiple interpretations/dimensions (definition, diagram, examples, graphical, textual)

4. Complex skills:

   For non-trivial problem solving tasks

   (a) analysis
      - measurement skills – what (variable isolation), how (precision, accuracy, error), expected worth/relevance (possible outcomes with implications), observation
      - data collection and analysis
      - classification problem solving
      - fault diagnosis
      - hypothesis formation, predicting
      - theory validation
   (b) synthesis

- system design
- programming/algorithm design
- repair, prescription
- theory formation
- organization/taxonomy creation – generalization, refinement
- creativity??

(c) transformation: (–of one problem or representation into another, such as transforming a word problem into an algebraic representation.)

Bibliography

[1] Abelson, H. (1982). *Apple LOGO*, Byte/McGraw-Hill, Peterborough, NJ.

[2] Abelson, H., diSessa, A., & Rudolph, L. (1974). "Velocity Space and the Geometry of Planetary Orbits," MIT AI Memo 320.

[3] Ableson,H. & diSessa, A. (1979). *Turtle Geometry*, MIT Press, Cambridge, MA.

[4] Adams, J. (1974). *Conceptual Blockbusting, A Pleasurable Guide to Better Problem Solving*, W. W. Norton & Co., New York.

[5] Alexander, et. al. (1986). "Knowledge Level Engineering: Ontological Analysis," *AAAI-86 proceedings.*

[6] Anderson, J, Farrell, R. & Sauers, R. (1984). "Learning to Program in Lisp," *Cognitive Science*, Vol. 8.

[7] Anderson, J. (1976). *Language, Memory, and Thought*, Erlbaum Assos., Hillsdale, NJ.

[8] Anderson, J. (1980). "A General Learning Theory and its Applications to the Acquisition of Proof Skills in Geometry," CMU tech. report 80-2.

[9] Anderson, J. (1983). *The Architecture of Cognition*, Harvard Univ. Press, Cambridge, MA.

[10] Anderson, J. (1984). "Cognitive Psychology and Intelligent Tutoring," draft paper.

[11] Anderson, J. (1985). "Skill Acquisition: Compilation of Week-Method Problem Solutions," ONR report ONR-85-1.

[12] Anderson, J. & Reiser, B. (1985). "The Lisp Tutor," *BYTE*, April 1985.

[13] Anderson, J. & Skwarecki, E. (1986). "The Automated Tutoring of Introductory Computer Programming," Communications of the ACM, Vol. 29 No. 9.

[14] Anderson, J., Boyle, F. & Reiser, B. (1985). "Intelligent Tutoring Systems," *Science*, Vol. 228.

[15] Anderson, J., Boyle, F. & Yost, G. (1985). "The Geometry Tutor," *IJCAI-85 proceedings.*

[16] Anderson, J., Boyle, F., Farrell, R. & Reiser, B. (1984). "Cognitive Principles in the Design of Computer Tutors," *Proceedings of Cognitive Science Society Conference.*

[17] Arons, A. (1984). "Computer-Based Instructional Dialogs in Science Courses," *Science,* Vol. 224(4653).

[18] Ashley, K. (1984). "Reasoning by Analogy: A Survey of Selected AI Research with Implications for Legal Expert Systems," *Proceedings of the First Ann. Conf. on Law and Technology.*

[19] Ashley, K. & Rissland, E. (1985). "Toward Modeling Legal Argument," aaa UMass Dept. of Computer and Information Science working paper.

[20] Barr, A. & Feigenbaum, E. (Eds.) (1982). "Applications- oriented AI Research: Education," *The Handbook of Artificial Intelligence, Volume 2*, William Kaufman, Inc., Los Altos.

[21] Barr, Cohen, & Feigenbaum (Eds.) (1982). *The Handbook of Artificial Intelligence, Volumes I, II, and III*, William Kaufman, Inc., Los Altos, CA.

[22] Bayman, P. & Mayer, R. (1983). "A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements," Communications of the ACM, Vol. 26 No. 9.

[23] Bobrow, D. et. al. (1986). "CommonLoops, Merging Lisp and Object-Oriented Programming," ACM, September 1996

[24] Bonar, J. (1982). "Natural Problem Solving Strategies and Programming Language Constructs," UMass Dept. of Computer and Information Science tech. report 82-11.

[25] Bonar, J. (1985). "Understanding the Bugs of Novice Programmers," UMass Dept. of Computer and Information Science tech. report 85-12.

[26] Bonar, J. & Cunningham, R. (1986). "Bridge: An Intelligent Tutor for Thinking About Programming-DRAFT paper," LRDC working paper.

[27] Bonar, J. & Soloway, E. (1983). "The Bridge From Non- Programmer to Programmer," Umass Dept. of Computer and Information Science tech. report 83-18.

[28] Bonar, J. & Soloway, E. (1985). "Pre-Programming Knowledge: A Major Source of Misconceptions in Novice Programmers," *Human-Computer Interaction*, Fall 1985.

[29] Bonar, J., Cunningham, R., & Schultz, J. (1986). "An Object-Oriented Architecture for Intelligent Tutoring Systems," *Proceedings of the First Annual Conf. on Object Oriented Programming, Systems, Languages, and Applications.*

[30] Bork, A. (1985). *Personal Computers for Education*, Harper & Row, Cambridge, MA.

[31] Brachman, R. & Levesque, H. (Eds.) (1985). *Readings in Knowledge Representation*, Morgan Kaufman Publ. Inc., Los Altos, CA.

[32] Brachman, R. & Schmoltze, J. (1985). "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science*, No. 9.

[33] Brown, D. (1986a). "The Role of Analogies, Metaphors, and Models in Assisting Comprehension," UMass Cognitive Processes Research Group working paper.

[34] Brown, D. (1986b). "Understanding Understanding Mechanical Systems in a Newtonian Framework," UMass Cognitive Processes Research Group working paper.

[35] Brown, D. & Clement, J. (1986a). "An Investigation of the Effectiveness of Logical and Causal Explanations for Overcoming Misconceptions in Mechanics," UMass Cognitive Processes Research Group working paper.

[36] Brown, D. & Clement, J. (1986b). "Overcoming Misconceptions in Mechanics: Theory and Practice–A Literature Review," UMass Cognitive Processes Research Group working paper.

[37] Brown, D., Clement, J., & Murray, T. (1985). "Tutoring Specifications for a Computer Program Which Uses Analogies to Teach Mechanics," UMass Cognitive Processes Research Group working paper.

[38] Brown, J. S. & deKleer, J. (1984). "A Framework for a Qualitative Physics," *Proceedings of Cognitive Science Society.*

[39] Brown, J. S., Burton, R. R., & deKleer, J. (1982). "Pedagogical, Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, and III," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[40] Brown, J. S., Burton, R., & Clancey, W. (1984). "Applications of Artificial Intelligence to Training and Education," *AAAI 1984 Conference Tutorial Program.*

[41] Bruner, J. (1966). *Toward a Theory of Instruction*, Harvard Univ. Press, Cambridge, MA.

[42] Burton, R. R. (1982). "Diagnosing Bugs in a Simple Procedural Skill," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[43] Burton, R. R., & Brown, J. S. (1982). "An Investigation of Computer Coaching for Informal Learning Activities," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[44] Carbonell, J. R. (1970). "AI and CAI; An Artificial Intelligence Approach to Computer Aided Instruction," *IEEE Transactions on MAN-Machine Systems*, MMS-11(4).

[45] Carr, B. & Goldstein, I. (1977). "Overlays: a Theory of Modelling for Computer Aided Instruction," MIT AI Memo 406.

[46] Chandrasekaran, B. (1985). "Generic Tasks in Expert System Design and Their Role in Explanation of Problem Solving," The Ohio State Univ., Dept. of Computer and Information Science, tech. report.

[47] Chandrasekaran, B. & Josephson, J. (1986). "Explanation, Problem Solving, and New Generation Tools: A Progress Report," The Ohio State Univ. Dept. of Computer and Information Science, tech. report.

[48] Charniac, E., & McDermott, D. (1985). *Introduction to Artificial Intelligence*, Addison-Wesley Publ. Co., Reading, MA.

[49] Chi, M., Feltovich, P. & Glaser, R. (1981). "Categorization and Representation of Physics Problems by Experts and Novices," *Cognitive Science*, (5).

[50] Clancey, W. (1982). "Tutoring Rules for Guiding a Case Method Dialogue," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[51] Clancey, W. (1984). "Teaching Classification Problem Solving," *Cognitive Science Socitey Proceedings.*

[52] Clancey, W. (1985). "Heuristic Classification," *Artificial Intelligence*, January, 1985.

[53] Clancey, W. (1986a). "Qualitative Student Models," *Annual Review of Computer Science*, February 1986.

[54] Clancey, W. (1986b). "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ONR Final Report 1979-1985," *The AI Magazine*, August, 1986.

[55] Claxton, G. (1985). "Teaching and Acquiring Scientific Knowledge," *Kelly in the Classroom: Educational Applications of Personal Construct Psychology*, T. Keen & M. Pope (Eds.), Cybersystems Inc., Montreal.

[56] Claxton, G. (1986). "Towards a Learning Theory for Science Teachers," *Children's Intuitive Scientific Ideas*, P. Block & A. Lowens (Eds.), Groom Helm Publ. Inc., .

[57] Clement, J. (1982). "Student's Preconceptions in Introductory Mechanics," *American Journal of Physics*, Vol. 50 No. 1.

[58] Clement, J. (1983). "A Conceptual Model Discussed by Galileo and Used Intuitively by Physics Students," *Mental Models*, Gentner & Stevens (Eds.), Lawrence Erlbaum, Hillsdale, NJ.

[59] Clement, J. (1986). "Methods for Evaluating the Validity of Hypothesized Analogies," *Cognitive Science Society proceedings.*

[60] Clement, J. & Brown, D. (1984a). "Finding 'Anchor Beliefs' for Grounding Instruction of Student's Intuitions," UMass Cognitive Processes Research Group working paper.

[61] Clement, J., & Brown, D. (1984b). "Using Analogical Reasoning to Deal With 'Deep' Misconceptions in Physics," UMass Cognitive Processes Research Group working paper.

[62] Clement, J., Lockhead, J. & Soloway, E. (1980). "Positive Effects of Computer Programming on the Student's Understanding of Variables and Equations," *Proceedings of the Assos. for Computing Machinery.*

[63] Cohen, P. (1984a). "Progress Report on a Theory of Endorsements: A Heuristic Approach to Reasoning About Uncertainty," UMass Dept. of Computer and Information Science Tech. Report 84-15.

[64] Cohen, P. (1984b). "Progress Report on the Theory of Endorsements: A Heuristic Approach to Reasoning About Uncertainty," UMass Dept. of Computer and Information Science tech. report 84-15.

[65] Cohen, P. (1985). "Numeric and Symbolic Reasoning About Uncertaity in Expert Systems," UMass Dept. of Computer and Information Science Tech. Report 84-25.

[66] Cohen, P., & Gruber, T. (1985). "Reasoning About Uncertainty: A Knowledge Representation Perspective," UMass Dept. of Computer and Information Science tech. report 85-24.

[67] Cohen, P., Day, D., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D. & Berman, P. (1986). "Management of Uncertainty in Medicine," UMass Dept. of Computer and Information Science working paper.

[68] Cohen, P., et. al. (1985). "Representativeness and Uncertainty in Classification Systems," UMass Dept. of Computer and Information Science tech. report 85-26.

[69] Collins, A. (1977). "Processes in Acquiring Knowledge," *Schooling and the Acquisition of Knowledge*, Anderson, Spiro, & Montague (Eds.), Lawrence Erlbaum Associates, Hillsdale, N. J..

[70] Collins, A. (1985). "Component Models of Physical Systems," *Proceedings of Cognitive Science Society.*

[71] Collins, G. (1985). "Teleology + Bugs = Explanations," *Cognitive Science Society conference proceedings.*

[72] Confrey, J. (1985). "A Constructivist View of Mathematics Instruction: A Theoretical Perspective," *Proceedings of the American Educational Research Association.*

[73] Cook, M. (1983). "An Intelligent Tutor for Plane Geometry," UMass Dept. of Computer and Information Science masters project computer program.

[74] Cunningham, P., Iberall, T., & Woolf, B. (1985). "CALEB: An Intelligent Second Language Tutor," UMass working paper.

[75] Davis, R. & Buchanan, B. (1977). "Meta-Level Knowledge: Overview and Applications," *IJCAI-77 Proceedings.*

[76] Davis, R., Buchanan, B., & Shortlife, E. (1977). "Production Rules as a Representation for a Knowledge-Based Consultant Program," *Artificial Intelligence*, 8(1).

[77] DeBono, E. (1970). *Lateral Thinking; A Textbook to Creativity*, Ward Lock Educational Ltd., London.

[78] Dede, C. (1985). "A Review and Synthesis of Recent Research in Intelligent Computer Assisted Instruction," To be published in the International Journal for Man-Machine Studies.

[79] Dede, C. (1986). "Artificial Intelligence Applications to High Technology Training," to be published in the Journal of Educational Communications and Technology.

[80] deKleer, J. & Williams, B. (1986). "Reasoning About Multiple Faults," *AAAI-86 proceedings.*

[81] deKleer, J., & Brown, J. S. (1980). "Mental Models of Physical Mechanisms," Xerox Palo Alto Report CSI-3.

[82] deKleer, J., & Brown, J. S. (1982). "Assumptions and Ambiguities in Mechanistic Mental Models," Xerox Palo Alto Tech. Report.

[83] diSessa, A. (1977). "On 'Learnable' Representations of Knowledge: A Meaning for the Computational Metaphor," MIT A.I. Lab. LOGO Memo No. 47.

[84] diSessa, A. (1979). "On Learning Representations of Knowledge: A Meaning for the Computational Metaphor," *Cognitive Process Instruction*, Lockhead & Clement (Eds.), The Franklin Institute Press, .

[85] diSessa, A. (1985). "Knowledge in Pieces," Address to the Fifteenth Symposium of the Jean Piaget Society.

[86] diSessa, A. & Ableson, H. (1986). "BOXER: A Reconstructible Computational Medium," Communications of the ACM, Vol. 29 No. 9.

[87] Driver, R. (1973). "The Representation of Conceptual Frameworks of Young Adolescent Science Students," Ph.D. dissertation, University of Illinois.

[88] Eisenberg, M. & Peelle, H. (1983). "APL Learning Bugs," Communications of the ACM 8/83.

[89] Elliot, P. (1978). "Computer 'Glass Boxes' as Advanced Organizers in Mathematics Instruction," Int. Jrnl. Math, Science, and Technology Education, Vol. 9 No. 1.

[90] Erman, L., Hayse-Roth, F., Lesser, V. & Reddy, D. (1980). "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computer Survey Report 84-21*, Vol. 12 N0. 2.

[91] Falkenhainer, B., Forbus, K., & Gentner, D. (1986). "The Structure-Mapping Engine," *AAAI-86 proceedings.*

[92] Farrell, R., Anderson, J., & Reiser, B. (1984). "An Interactive Computer-based Tutor for LISP," *AAAI-84 proceedings.*

[93] Flavell, J. (1980). "Speculations About the Nature and Development of Metacognition," *Metacognition, Motivation, and Learning*, Kluwe & Weinert (Eds.).

[94] Forbus, K. (1981). "A CONLAN Primer," BBN report No. 4491.

[95] Forbus, K. (1982a). "Spatial and Qualitative Aspects of Reasoning About Motion," MIT working paper.

[96] Forbus, K. (1982b). "Qualitative Reasoning About Physical Processes," MIT working paper.

[97] Forbus, K. (1983a). "Qualitative Reasoning About Space and Motion," *Mental Models*, Gentner & Stevens (Eds.), Earlbaum Asso., Hillsdale, NJ.

[98] Forbus, K. (1983b). "Measurement Interpretation in Qualitative Process Theory," *IJCAI-83 proceedings.*

[99] Forbus, K. (1984a). "Qualitative Process Theory," MIT Dissertation.

[100] Forbus, K. (1984b). "An Interactive Laboratory for Teaching Control System Concepts," BBN report No. 5511.

[101] Forbus, K. (1985). "The Problem of Existence," *Proceedings of the Cognitive Science Society Conference.*

[102] Forbus, K. & Gentner, D. (1984). "Learning Physical Domains: Towards a Theoretical Framework," *Machine Learning: Recent Progress*, Carbonell & Mitchell (Eds.), Tioga Press, .

[103] Forbus, K. & Gentner, D. (1986). "Causal Reasoning about Quantities," *Proceedings of the Cognitive Science Society.*

[104] Forbus, K. & Stevens, A. (1981). "Using Qualitative Simulation to Generate Explanations," BBN report No. 4490.

[105] Genesereth, M. R. (1982). "The Role of Plans in Intelligent Teaching Systems," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[106] Gentner, D. (1983). "Structure-Mapping: A Theoretical Framework for Analogy," *Cognitive Science*, Vol. 7 No. 2.

[107] Gentner, D. & Stevens, A. (Eds.) (1983). *Mental Models*, Lawrence Erlbaum Asso., Hillsdale, NJ.

[108] Gentner, D. & Toupin, C. (1986). "Systematicity and Surface Similarity in the Development of Analogy," *Cognitive Science*, Vol. 10.

[109] Ginsburg, H. & Opper, S. (1969). *Piaget's Theory of Intellectual Development, An Introduction*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

[110] Glaser, R. (1984). "Education and Thinking: The Role of Knowledge," *American Psychologist*, Vol. 39 No. 2.

[111] Goldberg, F. & McDermott, L. (1984). "Common Sense Knowledge Versus Formal Physics Knowledge in Geometrical Optics," presented at Amer. Assos. of Physics Teachers conference.

[112] Goldstein, I. P. (1982). "The Genetic Graph: A Representation of the Evolution of Procedural Knowledge," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[113] Gordon, W. (1961). *Synectics*, Harper & Row, New York.

[114] Grosz, B. (1986). "A Theory of Discourse Structure," *Proceedings of the Cognitive Science Society.*

[115] Grosz, B. & Sidner, C. (1985). "The Structure of Discourse Structure," *AAAI-85 Proceedings.*

[116] Hayse, P. (1985a). "The Second Naive Physics Manifesto," *Readings in Knowledge Representation*, Brachman & Levesque (Eds.), Morgan Kaufman, Los Altos.

[117] Hayse, P. (1985b). "The Logic of Frames," *Readings in Knowledge Representation*, Brachman & Levesque (Eds.), Morgan Kaufman Publishers, Inc., Los Altos, CA.

[118] Heller, H. & Reif, F. (1984). "Prescribing Effective Human Problem-Solving Processes: Problem Description in Physics," *Cognition and Instruction*, 1 (2).

[119] Horowitz, P. & White, B. (1986). "Thinkertools Annual Progress Report," Bolt Beranek and Newman Technical Report.

[120] Horvitz, E., Heckerman, D., & Langlotz, C. (1986). "A Framework for Comparing Alternative Formalisms for Plausable Reasoning," *AAAI-86 proceedings.*

[121] Joyce, B. & Well, M. (1972). *Models of Teaching*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

[122] Kimbal, R. (1982). "A Self-improving Tutor for Symbolic Integration," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[123] Kulik, C. & Kulik, J. (1986). "Effectiveness of Computer- Based Education in Colleges," to appear in AEDS Journal.

[124] Kulik, C., Kulik, J., & Shwalb, B. (1986). "The Effectiveness of Computer-based Adult Education: A Meta-analysis," Jrnl. Educational Computing Research, Vol. 2(2).

[125] Kulik, J. (1985). "Consistencies in Findings on Computer- Based Education," presented to the annual meeting of the American Educational Research Association.

[126] Lakatos, I. (1976). *Proofs and Refutations, The Logic of Mathematical Discovery*, Cambridge Univ. Press, Cambridge, MA.

[127] Larkin, J. (1980). "Spatial Reasoning in Solving Physics Problems," CMU working paper.

[128] Larkin, J. (1983a). "A General Knowledge Structure for Learning or Teaching Science," *Classroom Computers and Cognitive Science*, Wilkinson (Eds.), Academic Press, .

[129] Larkin, J. (1983b). "The Role of Problem Representation in Physics," *Mental Models*, Gentner & Stevens (Eds.), Erlbaum Asso., Hillsdale, NJ.

[130] Lawson, A. & Wollman, W. (1975). "Physics Problems and the Process of Self-Regulation," *The Physics Teacher*, Nov. 1975.

[131] Leant, D. (1983). "On Automated Scientific Theory Formation: A Case Study Using the AM Program," *Machine Intelligence*, VOl. 9.

[132] Leiberman, H. (1980). "There's More to Menu Systems Than Meets the Screen," MIT tech. report.

[133] Leiberman, H. (1985). "An Example Based Environment for Beginning Programmers (Draft)," MIT report.

[134] Lesser, V. (1984). "Control in Complex Knowledge-based Systems," Tutorial at the IEEE Computer Society AI conference.

[135] Littman, D., Pinto, J., & Soloway, E. (1986). "An Analysis of Tutorial Reasoning About Programming," *AAAI-86 proceedings.*

[136] Lockhead, J. & Clement, J. (Eds.) (1979). *Cognitive Process Instruction: Research on Teaching Thinking Skills*, The Franklin Institute Press, .

[137] London, B. (1986). "Integrated, Multiple Viewpoints in Diagnostic Student Modelling," *submitted to AAAI-86.*

[138] Malone, T. & Levin, J. (Eds.) (1981). "Microcomputers in Education: Cognitive and Social Design Principles," working paper.

[139] Mats, M. (1982). "Towards a Process Model for High School Algebra Errors," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[140] McCloskey, M. (1983). "Intuitive Physics," *Scientific American*, Vol. 48.

[141] McDermott, L. (1984). "Research on Conceptual Understanding in Mechanics," *Physics Today*, July 1984.

[142] McDonald, D., Brooks, J., Woolf, B., & Werner, P. (1986). "Transition Networks for Discourse Management," UMass Dept. of Computer and Information Science working paper.

[143] Michener, E. R. (1978). "Understanding Understanding Mathematics," *Cognitive Science*, Vol. 2, No. 4.

[144] Miller, M. L. (1982). "A Structured Planning and Debugging Environment for Elementary Programming," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[145] Minsky, M. (1981). "A Framework for Representing Knowledge," *Mind Design*, J. Haugeland (Eds.), MIT Press, Cambridge, MA.

[146] Murray, T. & Woolf, B. (1986). "Knowledge Representation in a Physics Tutor," UMass Dept. of Computer and Information Science technical report 86-37.

[147] Murray, T., Clement, J., & Brown, D. (1985). "A Computer Program which Utilizes a 'Bridging Analogies' Tutoring Strategy for Dealing with Misconceptions in Mechanics," Cognitive Processes Research Group Tech. Report.

[148] Murray, T., Woolf, B., Brown, D., Clement, J., & Schultz. K. (1986). "Tutorial Strategies for Computers which Deal with Misconceptions in Physics," submitted to Third Int. Conf. on AI and Education.

[149] Nickerson, R., Perkins, D. & Smith, E. (1985). *The Teaching of Thinking*, Lawrence Erlbaum Asso., Hillsdale, NJ.

[150] Nii, H. Penny (1986). "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *The AI Magazine*, Summer-86.

[151] Novak, G. (1977). "Representations of Knowledge in a Program for Solving Physics Problems," *IJCAI-77 Proceedings.*

[152] Novak, G. & Araya, A. (1980). "Research on Expert Problem Solving in Physics," *Proceedings of the First National Conf. on AI.*

[153] O'Shea, T. (1982). "A Self-improving Quadratic Tutor," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[154] O'Shea, T. & Self, J. (1983). *Learning and Teaching With Computers: Artificial Intelligence in Education*, Prentice-Hall Inc., Englewood Cliffs, N.J..

[155] Osborne, R. (1984). "Children's Dynamics," *The Physics Teacher*, Nov. 1984.

[156] Palies, O., Caillot, M., Cauzinille-Marmeche, E., Lauriere, J., & Mathieu, J. (1980). "Student Modelling by a Knowledge-based System," to be in Computational Intelligence, AI and Education issue.

[157] Papert, S. (1980). *MindStorms–Children, Computers, and Powerful Ideas*, Basic Books, Inc., New York.

[158] Park, O., Perez, R., and Seidel, R. (1986). "Intelligent CAI: Old Wine in New Bottles - Or, Is It Just Vinegar?," *Artificial Intelligence and Instruction: Application and Methods*, Kearsley (Eds.), Addison-Wesley, Reading, MA.

[159] Pea, R. (1983). "LOGO Programming and Problem Solving," Bank Street College tech. report 12, April 1983.

[160] Peelle, H. A. (1982). "Computer Vitamins," *Educational Computing Magazine*, September 1982.

[161] Peelle, H. A. (1986). "What is 'APL Thinking'?," *Proceedings of the APL86 conference.*

[162] Peelle, H. A. & Eisenberg, M. (1983). "APL Teaching Bugs," UMass School of Education working paper.

[163] Perkins, D. (1986). *Knowledge as Design*, Lawrence Erlbaum Asso., Hillsdale, NJ.

[164] Piaget, J. (1972). *The Principles of Genetic Epistemology*, Basic Books, NY.

[165] Polya, G. (1973). *How To Solve It*, Princeton University Press, New Jersey.

[166] Reif, F. & St. John, M. (1979). "Teaching Physicists' Thinking Skills in the Laboratory," *Amer. J. Phys.*, 47(11), Nov..

[167] Reisberg, D. & Chambers, D. (1986). "Neither Pictures Nor Propositions: The Intentionality of Mental Imagery," *Cognitive Science Society Proceedings.*

[168] Reiser, B., Anderson, J., & Farrell, R. (1985). "Dynamic Student Modeling in an Intelligent Tutor for Lisp Programming," *IJCAI-85 Proceedings.*

[169] Richer, M & Clancey, W. (1985). "Guidon-Watch: A Graphic Interface for Viewing a Knowledge-Based System," IEEE CG&A, November, 1985.

[170] Rissland, E. (1978). "Understanding Understanding Mathematics," *Cognitive Science*, Vol. 2 No. 4.

[171] Rissland, E. (1980). "The Structure of Knowleage in Complex Domains," UMass Dept. of Computer and Information Science tech. report 80-07.

[172] Rissland, E. (1981). "Constrained Example Generation," UMass Dept. of Computer and Information Science tech. report 81-24.

[173] Rissland, E. (1982). "Examples in the Legal Domain: Hypotheticals in Contract Law," *Proceedings of Cognitive Science Society.*

[174] Rissland, E. (1983). "A.I. and the Learning of Mathematics," UMass Dept. of Computer and Information Science tech. report 83-40.

[175] Rissland, E. (1984a). "Argument Moves in Hypotheticals," *Proceedings of the First An. Conf. on Law and Technology.*

[176] Rissland, E. (1984b). "Learning How to Argue: Using Hypotheticals," UMass Dept. of Computer and Information Science working paper.

[177] Rissland, E. & Ashley, K. (1980). "Hypotheticals as Heuristic Device," working paper.

[178] Rissland, E. R., Valcarce, E., & Ashley, K. (1984). "Explaining and Arguing With Examples," *AAAI-84 Proceedings.*

[179] Sacerdoti, E. (1974). "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, 5(2).

[180] Samples, B. (1978). *The Metaphoric Mind–A Celebration of Creative Consciousness*, Addison-Wesley, Reading, MA.

[181] Schank, R., & Childers, P. (1984). *The Cognitive Computer, On Language, Learning, and Artificial Intelligence*, Addison- Wesley Publ. Co., Reading, MA.

[182] Schank, R., & Riesbeck, C. (1981). *Inside Computer Understanding*, Lawrence Erlbaum Asso., Publ., Hillsdale, NJ.

[183] Schoenfeld, A. (1983). "Metacognition and Epistemological Issues in Mathematical Understanding," Univ. of Rochester working paper.

[184] Schwartz, J. (1980?). "The Geometric Supposer," (Software for APPLE computers).

[185] Schwebel, M. & Raph, J. (1973). *Piaget in the Classroom*, Basic Books, Inc., New York.

[186] ServiB, D. & Woolf, B. (1986). "E.T.: An Architecture for Error-Triggered Remediation In Intelligent Tutoring Systems," Presented at the 1987 AI in Education conference.

[187] Shapiro, S. & Rapaport, W. (1986). "SNePS Considered as a Fully Intentional Propositional Semantic Network," *AAAI-86 proceedings.*

[188] Shavlik, J. & Dejong, G. (1985). "Building a Computer Model of Learning Classical Mechanics," *Proceedings of the Cognitive Science Society.*

[189] Sherwood, B. & Kane, D. (1980). "A Computer-based Course in Classical Mechanics," *Computers and Education*, Vol. 4, Pergamon Press Ltd, .

[190] Shute, V. & Bonar, J. (1986). "Intelligent Tutoring Systems for Scientific Inquiry Skills," *Proceedings of the Cognitive Science Society.*

[191] Simpson, R. (1985). "A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation," Ph.D. Thesis, School of Info. & Computer Sci., Georgia Inst. of Technol..

[192] Slater, J., Petrossian, R., & Shyam-Sunder, S. (1985). "An Expert Tutor of Rigid Body Mechanics: Athena Cats - MACAVITY," IEEE.

[193] Sleeman, D. (1982). "Assessing Aspects of Competence in Basic Algebra," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[194] Sleeman, D., & Hendley, R. J. (1982). "ACE: A System Which Analyses Complex Explanations," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[195] Sleeman, D., Brown, J. S. (Eds.) (1985). *Intelligent Tutoring Systems*, Academic Press, New York, NY.

[196] Soloway, E. (1986). "Learning to Program = Learning to Construct Mechanisms and Explanations," Communications of the ACM, Vol. 29 No. 9.

[197] Soloway, E. & Johnson, L. (1984). "Remembrance of Blunders Past: A Retrospective on the Development of Proust," *Proceedings of the Cognitive Science Society.*

[198] Soloway, E., Woolf, B., Rubin, E., & Boarth, B. (1981). "MENO-II: An Intelligent Tutoring System for Novice Programers" IJCAI-81 Proceedings.

[199] Solso, R. (1979). *Cognitive Psychology*, Harcourt Brace Jovanovich, Inc., New York.

[200] Steele, G. (1984). *Common LISP, The Language*, Digital Press, .

[201] Stefik, M. & Bobrow, D. (1986). "Object Oriented Programming: Themes and Variations," *The AI Magazine*, Winter-86.

[202] Stevens, A. & Collins, A. (1977). "The Goal Structure of a Socratic Tutor," BBN Report No. 3518.

[203] Stevens, A., Collins, A., & Goldin, S. E. (1982). "Misconceptions in Students' Understanding," *Intelligent Tutoring Systems*, Sleeman & Brown (Eds.), Academic Press, New York, NY.

[204] Sullivan, M. & Cohen, P. (1984). "An Endorsement-Based Plan Recognition Program," UMass Dept. of Computer and Information Science tech. report 84-33.

[205] Tennyson, R. & Park, O. (1980). "The Teaching of Concepts: A Review of Instructional Design Research Literature," *Review of Educational Research*, Vol. 50 No. 1.

[206] Tversky, A. & Kahneman, D. (1974). "Judgement Under Uncertainty: Heuristics and Biases," Science, Vol. 185.

[207] Van Lehn, K. (1983). "Felicity Conditions for Human Skill Acquisition: Validating an AI-Based Theory," Xerox Palo Alto Research Center Technical Report CIS-21.

[208] Varma, V. & Williams, P. (Eds.) (1976). *Piaget, Psychology and Education*, F. E. Peacock Publ., Inc., Itasca, IL.

[209] Wenger, E. (1986). *Knowledge Communication Systems*, to be publ. by Morgan & Kaufman, Inc., Los Altos, CA.

[210] Weyer, S. (1982). "Searching for Information in a Dynamic Book," Xerox Palo Alto tech. report no. SCG-82-1.

[211] Whimbley, A. & Lockhead, J. (1982). *Problem Solving and Comprehension*, The Franklin Institute Press, Philadelphia.

[212] White, B. (1984). "Designing Computer Games to Help Physics Students Understand Newton's Laws of Motion," *Cognition and Instruction*.

[213] White, B. & Frederiksen, J. (1986a). "Intelligent Tutoring Systems Based Upon Qualitative Model Evolution," *AAAI-86 Conference Proceedings*.

[214] White, B. & Frederiksen, J. (1986b). "Progressions of Qualitative Models as a Foundation for Intelligent Learning Environments," BBN report No. 6277.

[215] White, B. & Frederiksen, J. (1986c). "Qualitative Models and Intelligent Learning Environmetns," *to appear in AI and Education*, Lawler and Yazdani (Eds.), Ablex Publ. Corp., .

[216] Winston, P. H. (1984a). *LISP*, Addison-Wesley Publ. Co., Reading, MA.

[217] Winston, P. H. (1984b). *Artificial Intelligence*, Addison Wesley Publ. Co., Reading, MA.

[218] Woods, W. (1970). "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, Vol. 13, No. 10.

[219] Woolf, B. (1984). "Context Dependent Planning in a Machine Tutor," UMass Dept. of Computer and Information Science technical report 84-21.

[220] Woolf, B., Blegen, D., Verloop, A. & Jansen, J. (1986). "Tutoring a Complex Industrial Process," *AAAI-86 Proceedings*.

[221] Woolf, B. & McDonald, D. (1983). "Human-Computer Discourse in the Design of a PASCAL Tutor," Umass Dept. of Computer and Information Science tech. report 83-27.

[222] Woolf, B. & McDonald, D. (1984). "Building a Computer Tutor: Design Issues," *IEEE Computer*, September 1984.

[223] Woolf, B. & McDonald, D. (1985). "Understanding Discourse Conventions in Tutoring," UMass Dept. of Computer and Information Science tech. report 85-22.

[224] Woolf, B. & Murray, T. (1987). "A Framework for Representing Tutorial Discourse," IJCAI-87.

[225] Woolf, B., Pustejovsky, J., McDonald, D. & Lander, S. (1980). "Discourse Processing in a Knowledge Acquisition Dialog," UMass Dept. of Computer and Information Science working paper.

[226] Xerox Special Information Systems, Vista Laboratory (1985). "CALLE Project Report," .

[227] Yankelovich, N., Meyorowitz, N., & van Dam, A. (1985). "Reading and Writing the Electronic Book," *Computer*, October, 1985.

[228] Yazdani, M. (1986). "Intelligent Tutoring Systems: An Overview," *Expert Systems*, Vol. 3. No. 3.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.